1. CSCI-604 Advanced Object-Oriented Programming Concepts

July/2015

Note: I will update the notes on a regular basis.
2. General Information
Course Title: CSCI-605 Advanced Object-Oriented Programming Concepts
Instructor: Hans-Peter Bischof
Office: 70-3005
Telephone: (585) 475-5568
Office Hours: [Tues|Thurs]day: 10am - noon, and by appointment
Course Home Page:
CS Home Page:
2.1. Topics

- 1 - Inheritance
- 1 - Generics
- 1 - Wildcards
- 2 - I/O: Files and Streams
- 2 - Threads
- 3 - Networking
- 3 - Collections
2.2. Grading
93-100% A
90-92% A-
87-89% B+
83-86% B
80-82% B-
77-79% C+
73-76% C
70-72% C-
60-69% D
Below 60% F

2.3. Tentative Schedule
Exam 1: jan/13/2016
Exam 2: jan/22/2016 - I might have to move this one
I will use concepts from the One part of each homework will use concepts not covered in class. I will point where you can find these materials. The first example is hw 1.[23].

2.4. Syllabus

2.5. Academic Honesty
— Academic Integrity
— You may help each other freely to complete homeworkâs as the purpose of the homeworkâs is to increase your understanding.
— This does not mean that someone else can do your homework for you. Any homework you submit must contain your significant intellectual contribution.
— The corollary is that you may not do someone else’s work for them either. A willing supplier of the material is as guilty of academic dishonesty as the receiver.
— Any help you receive from someone must be acknowledged in the work submitted. Failure to acknowledge the source of a significant idea or approach is considered plagiarism and not allowed.

Academic dishonesty is dealt with severely:
— You will receive a grade of F for the course.
— A note describing the details of your case will become part of your academic record.
— Repeated offenses or more serious violations may result in your being suspended or
— Violations of the Academic Integrity can also result in suspension, expulsion and even criminal charges.
2.6. Course Goals

- Intro into the language Java (1.7)
- Use of the major classes, like Collection Framework, RMI, Swing, etc.
- Introduction to design patterns.

This is not a “programming” course, per se. Programming is a means to an end, not an end in and of itself.
2.7. Web Resources

Example:

```java
public class HelloGoodbye {
    public static void main(String[] args) {
        System.out.println("Hello world");
        Runtime.getRuntime().addShutdownHook(
            new Thread() {
                public void run() {
                    System.out.println("Goodbye world");
                }
            });
        System.exit(0);
    }
}
```

Source Code: Src/1/HelloGoodbye.java

2.8. Texts

There is an almost infinite number of (not necessarily good) books about Java and even more about the World Wide Web. The following books I found quite useful, however.

<table>
<thead>
<tr>
<th>Author(s)</th>
<th>Title</th>
<th>Publisher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bruce Eckel</td>
<td>Thinking in Java</td>
<td>mindview.net/Books</td>
</tr>
<tr>
<td>Java Tutorial</td>
<td>java.sun.com/docs/books/tutorial/index.html</td>
<td>Java Tutorial (2nd Ed.)</td>
</tr>
<tr>
<td>Online Courses</td>
<td>java.sun.com/developer/onlineTraining/ Online Courses</td>
<td>Online Courses</td>
</tr>
<tr>
<td>Mughal/Rasmussen</td>
<td>0-201-59614-8</td>
<td>Mughal/Rasmussen</td>
</tr>
<tr>
<td>Kalin</td>
<td>0-13-019859-5</td>
<td>Kalin</td>
</tr>
<tr>
<td>Flanagan</td>
<td>1-56592-262-X</td>
<td>Flanagan</td>
</tr>
<tr>
<td>Flanagan</td>
<td>1-56592-371-5</td>
<td>Flanagan</td>
</tr>
</tbody>
</table>

Useful is also *The Java Language Specification*:

The following books were written by the Java developers themselves and are relatively useful. Second editions are available or in the making, more or less face-lifted for Java 1.2[3]:

<table>
<thead>
<tr>
<th>Author(s)</th>
<th>Title</th>
<th>Publisher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arnold/Gosling</td>
<td>0-201-31006-6</td>
<td>Arnold/Gosling</td>
</tr>
<tr>
<td>Campione/Walrath</td>
<td>0-201-31007-4</td>
<td>Campione/Walrath</td>
</tr>
<tr>
<td>Chan</td>
<td>0-201-37967-8</td>
<td>Chan</td>
</tr>
<tr>
<td>Chan/Lee</td>
<td>0-201-63458-9</td>
<td>Chan/Lee</td>
</tr>
<tr>
<td>Chan/Lee/Kramer</td>
<td>0-201-31002-3</td>
<td>Chan/Lee/Kramer</td>
</tr>
<tr>
<td>Chan/Lee</td>
<td>0-201-31003-1</td>
<td>Chan/Lee</td>
</tr>
<tr>
<td>Gosling/Yellin et al.</td>
<td>3-8273-1040-7</td>
<td>Gosling/Yellin et al.</td>
</tr>
<tr>
<td>Gosling/Yellin et al.</td>
<td>3-8273-1084-9</td>
<td>Gosling/Yellin et al.</td>
</tr>
<tr>
<td>Hamilton/Cattell/Fisher</td>
<td>0-201-30995-5</td>
<td>Hamilton/Cattell/Fisher</td>
</tr>
<tr>
<td>Kanerva</td>
<td>0-201-63456-2</td>
<td>Kanerva</td>
</tr>
<tr>
<td>Lea</td>
<td>0-201-69581-2</td>
<td>Lea</td>
</tr>
<tr>
<td>Liang/Sterns</td>
<td>0-201-32577-2</td>
<td>Liang/Sterns</td>
</tr>
<tr>
<td>Lindholm/Yellin</td>
<td>0-201-63452-X</td>
<td>Lindholm/Yellin</td>
</tr>
<tr>
<td>Sowizral/Rushforth et.al.</td>
<td>0-201-32576-4</td>
<td>Sowizral/Rushforth et.al.</td>
</tr>
<tr>
<td>Sridharan</td>
<td>0-13-749136-0</td>
<td>Sridharan</td>
</tr>
</tbody>
</table>

If you would like to take a Java exam to become a Sun Certified Programmer for the Java 2 Platform, here is an example:
2.9. Environment
Java 1.7 will be used

2.10. Homeworks
The solutions for the homeworks are in Java 1.7.
The homeworks are done in teams of two. The team has to meet with a grader. Each team member must be able to explain the solution to her/him. The grade for each project is based on the correctness, your explanation, and the quality of the code.
A solution must be submitted by each student for the first homework.
You can submit as often as you like, but only the last submission counts.
Signup sheets will be up at the graduate lab door by Wednesday. Be on time. You might receive 0 points for the homework if you are late.
2.11. Other Things

Course Organization

• cheating
• lectures - ask questions
• homework
  — Grader
  — Individual questions
2.12. Object-Oriented Ingredients

- Objects
- Class
- Methods
- Encapsulation
- Inheritance
- Polymorphism

2.13. Object

Objects are the things you think about first in designing a program and they are also the units of code that are eventually derived from the process.

In between, each object is made into a generic class of object and even more generic classes are defined so that objects can share models and reuse the class definitions in their code.

Each object is an instance of a particular class or subclass with the class’s own methods or procedures and data variables. An object is what actually runs in the computer.

2.14. Class

A class consists of all objects with like state which know exactly the same methods, i.e., a class knows the structure of its objects and contains the methods to manipulate the structure. An object is called an instance of a class.

Given a class, one normally creates objects. Objects are created dynamically with the prefix operator new which in turn calls a constructor method to initialize the instance variables. Uninitialized instance variables are zeroed.

Methods mostly access the instance variables of the receiver. If methods return their receiving objects, method calls (messages) can be cascaded.

The class is one of the defining ideas of object-oriented programming. Among the important ideas about classes are:

2.15. Encapsulation

Encapsulation is the inclusion within a program object of all the resources need for the object to function — basically, the methods and the data.

Other objects adhere to use the object without having to be concerned with how the object accomplishes it. The idea is "don’t tell me how you do it; just do it." An object can be thought of as a self-contained atom. The object interface consists of public methods and instance data.

2.16. Methods

A method is a programmed procedure that is defined as part of a class and included in any object of that class. A class (and thus an object) can have more than one method. A method in an object can only have access to the data known to that object, which ensures data integrity among the set of objects in an application. A method can be re-used in multiple objects.

- A class can have subclasses that can inherit all or some of the characteristics of the class. In relation to each subclass, the class becomes the superclass.
Subclasses can also define their own methods and variables that are not part of their superclass. The structure of a class and its subclasses is called the class hierarchy.

Question: What is the difference between class and object

2.17. Inheritance

Inheritance is the concept that when a class of objects is defined, any subclass that is defined can inherit the definitions of one or more general classes.

This means for the programmer that an object in a subclass need not carry its own definition of data and methods that are generic to the class (or classes) of which it is a part. This not only speeds up program development; it also ensures an inherent validity to the defined subclass object (what works and is consistent about the class will also work for the subclass)

2.18. Polymorphism

Polymorphism (from the Greek meaning "having multiple forms") is the characteristic of being able to assign a different meaning to a particular symbol or "operator" in different contexts.

2.19. Object-Oriented Programming Keywords

- Object
- Class
- Encapsulation
- Methods
- Encapsulation
- Methods
- Inheritance
- Polymorphism
- Reuse
3. What is Java?

Thanks to Java is a new programming language developed at Sun under the direction of James Gosling. As far as possible it is based on concepts from C, Objective C and C++.

Java is interpreted and loads classes dynamically. There are CPU chips for Java; Sun showed a prototype Java computer early in 1996 and by now it is commercially available (if slow).

HotJava is a Web browser, that was implemented in Java using a modular architecture. It loads protocol modules and other applications (applets) dynamically from the local system or over a network and thus adapts to new tasks.

According to Hoff, Shaio and Starbuck, Java is “simple, object oriented, statically typed, compiled, architecture neutral, multi-threaded, garbage collected, robust, secure, extensible and well understood. Above all, however, Java is fun!”

These terms are elaborated on below -- I do not quite accept all these claims, however ...

simple
Java is related to languages like C or C++, but to make the language small and easy to learn all inessentials were eliminated. This leads to more uniform programs and simpler maintenance.

Unfortunately, Java 1.1 introduced significant new, useful, but different concepts. There is a proliferation of libraries as various companies try to turn individuality into market dominance.
**object-oriented**
Modern programs are distributed and have graphical user interfaces. Both can be implemented more easily with object orientation. Unlike C++, Java is fully object oriented and thus furthers the right programming style for these problem areas.

**statically typed**
All data must be declared with types so that data and operations may be matched as far as possible during compilation. Methods are dynamically bound but overloaded signatures are decided during compilation. Like Objective C, Java permits type queries and object analysis, e.g., for the existence of methods, using the package java/lang/reflect at runtime.

**compiled**
Unlike TCL, Java avoids repeated source analysis. A program is compiled into byte codes, the machine language of the Java Virtual Machine (JVM). The JVM is interpreted or compiled just in time. Classes and methods are bound symbolically, i.e., classes can be recompiled individually.

**architecture neutral**
Byte codes are the same everywhere; only the JVM has to be implemented for a new platform. Java programs should run everywhere and can be distributed as binaries. Unlike C and C++, Java completely specifies the capacity and behavior of the primitive data types thus eliminating a serious portability problem. Swing is implemented without any native code, relying only on the API defined in JDK 1.1.
multi-threaded
Graphical user interfaces provide the illusion of parallel execution. Threads offer an elegant implementation. Java has a thread system based on classes and the language contains simple synchronization mechanisms (monitors). Many class libraries are thread-safe.

garbage collected
Dynamic memory management as in C and C++, where the programmer attends to reusing resources, is efficient but error prone. Java only knows dynamic objects and vectors and completely frees the programmer from the problem of memory reuse. Garbage collection runs as a parallel thread and thus should not be a bottleneck in critical situations.

robust
Exceptions are an integral part of Java for error handling. The programmer is constantly forced to consider error possibilities in libraries and the compiler can check that exceptions are not hidden or overlooked.

secure
An interpreter can pretty much ensure that the interpreted program cannot crash it’s platform. In connection with the Web, Java has additional security mechanisms that constrain foreign programs so that viruses are considered impossible — in spite of the fact that binary Java programs can run on arbitrary platforms. Various groups have demonstrated, however, that security holes do exist. There is a move toward digitally signed programs and distributed trust.

extensible
Java methods can be implemented in other languages using the Java Native Interface (JNI). In principle, arbitrary libraries can be accessed as long as other security or portability aspects do not prevail.

well understood
While Java is a new language it’s concepts are well known. The language definition is comprehensive and still short. Unlike C++, a programmer can certainly understand Java completely and use all of it. Java’s relationship to C and it’s dialects makes it easy to get into the language, although there are a number of subtle differences to be aware of.
3.1. Different JVMS ...

Java Standard Edition (J2SE)
Java 2 Micro Edition (J2ME), Connected Device Configuration (CDC)
Java 2 Micro Edition (J2ME), Connected Limited Device Configuration (CLDC)

<table>
<thead>
<tr>
<th>Edition</th>
<th>Configuration</th>
<th>RAM</th>
<th>Typical Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>J2ME</td>
<td>CLDC</td>
<td>32k</td>
<td>Cellphone, pagers</td>
</tr>
<tr>
<td>J2ME</td>
<td>CDC</td>
<td>512k</td>
<td>PDA's</td>
</tr>
<tr>
<td>J2SE</td>
<td>CDC</td>
<td>Lots</td>
<td>Desktop applications</td>
</tr>
</tbody>
</table>

3.2. J2SE Platform at a Glance

Copied from

3.3. The first Program: Hello.java

```java
1     /**
2     * Classical: Hello, World
3     *
4     *
5     * @version $Id: Hello.java,v 1.3 2001/06/06 23:05:46 hpb Exp hpb $*
6     *
7     * @author hpb
8     *
9     * Revisions:
10     *
11     * Revision 1.41 2013/06/06 16:19:12 hpb
12     * Revision 1.42 2014/08/06 10:31:21 hpb
13     * Initial revision
14     *
15     */
16
17     class Hello {
18          public static void main (String args []) { // main program
19              System.out.println("Hello World!");
20          }
21     }
22
Source Code: Src/3/Hello.java
```

% javac Hello.java
% java Hello
Hello, World

Please take a look at the
Please take a look into how to use
### 3.4. The first Application

Hello outputs a text that is specified in the program.

Hello demonstrates how to code a minimal application.

An application is a stand-alone Java program with a graphical user interface, usually based on the Abstract Window Toolkit.

```java
import javax.swing.*;
import java.awt.event.*; // for WindowEvent

public class HelloWorld {

    public static void main(String[] args) {
        JFrame frame = new JFrame("HelloWorld");
        final JLabel label = new JLabel("Hello World");
        frame.getContentPane().add(label);

        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });

        frame.pack();
        frame.setVisible(true);
    }
}

Source Code: Src/3/HelloWorld.java
```
3.5. The first applet -- applets/hello

Hello outputs a text that is specified in the program.
Hello is an applet and demonstrates how Java is used in Web documents.

```java
/**
 * Classical Applet: Hello, World
 * @version $Id$
 * @author hpb
 * Revisions:
 * $Log$
 */
import java.applet.*; // find Applet class
import java.awt.*; // find Graphics class
import javax.swing.*; // find Swing class

/** A class with the first Java applet */
public class HelloApplet extends JApplet {

    /** hook to initialize the applet */
    public void init () {
        resize(150, 25);
    }

    /** redraw the Applet */
    public void paint (Graphics g) {
        g.drawString("Hello, World", 50, 25);
    }
}

Source Code: Src/3/HelloApplet.java
```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html><head>
<title></title>
<style type="text/css">
<!--code { font-family: Courier New, Courier; font-size: 10pt; margin: 0px; }-->
</style>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head><body>

<!-- ============================================================== -->
<!-- =Java Sourcecode to HTML automatically converted code = -->
<!-- =Java2Html Converter 5.0 [2006-02-26] by Markus Gebhard markus@jave.de = -->
<!-- =Further information: http://www.java2html.de = -->
<div align="left" class="java">
<table border="0" cellpaddings="3" cellspacing="0" bgcolor="#ffffff">
<tr>
<!-- start source code -->
<font color="#3f5fbf">/**

* Classical: Hello, World

* @author hpb

* Revisions:

* Revision 1.1 2003/06/06 16:19:12 hpb

* Initial revision

*/

class Hello {

public static main(String system.out.println("Hello World!"));

}</font>
3.6. Documentation — javadoc

The various JDK packages are documented on HTML pages in a standardized format that contains very many references to other classes or replaced methods etc.

See here:

creates the documentation directly from the program sources. Special comments /** ... */ before class, variable and method declarations are transferred to the documentation. The comments may contain significant tags

See coding standard
4. Java Programming Basics

- Classpath:
  - The JVM is using a class loader to load classes used by an application on an as-needed basis.
  - The CLASSPATH environment variable tells the class loader where to find the classes.
  - Classpath entries can be directories that contain class files for classes, or archive files (such as .zip or .jar files) that contain classes.
  - Classpath entries are colon-separated on Unix-type systems and semicolon-separated on MS Windows systems.
  - In a bash/sh environment:
    ```
    % CLASSPATH=/home/hpb/Jrms/classes.jar:$CLASSPATH
    % export CLASSPATH
    ```
  - For javac/java/... on a UNIX platform:
    ```
    % jdkTool -classpath path1:path2...
    ```
  - Which classes are used?
    ```
    % java -verbose Hello.java
    java -verbose Hello.java
    [Opened /usr/local/j2sdk1.5.0/jre/lib/rt.jar]
    [Opened /usr/local/j2sdk1.5.0/jre/lib/jsse.jar]
    [Opened /usr/local/j2sdk1.5.0/jre/lib/jce.jar]
    [Opened /usr/local/j2sdk1.5.0/jre/lib/charsets.jar]
    [Loaded java.lang.Object from /usr/local/j2sdk1.5.0/jre/lib/rt.jar]
    ...
    ```

- Coding Standard: See

4.1. Comment

See also

Java defines three kinds of comments:

```
/* text */
```

A traditional comment: all the text from the ASCII characters /* to the ASCII characters */ is ignored (as in C and C++).

```
// text
```

A single-line comment: all the text from the ASCII characters // to the end of the line is ignored (as in C++).

```
/** documentation */
```

A documentation comment: the text enclosed by the ASCII characters /** and */ can be processed by a separate tool to prepare automatically generated documentation of the following class, interface, constructor, or member (method or field) declaration.
4.2. Identifier

An is an unlimited-length sequence of Java letters and Java digits, the first of which must be a Java letter. An identifier cannot have the same spelling as a keyword, Boolean literal, or the null literal.

1. Can be any length
2. First character must be a letter \{ 0, .. 9 \}
3. Following characters can be letters or digits
4. Are case sensitive: TOM is NOT the same as Tom
5. A Java reserved word CANNOT be used as an identifier, ex. true
6. Must be declared to be of some type.

Identifier should/must be self explained. Please do the following

The words i, k, lth, ... have no real meaning for a human being.

<table>
<thead>
<tr>
<th>lessThanTen</th>
<th>versus</th>
<th>ltt</th>
</tr>
</thead>
<tbody>
<tr>
<td>thisIsSorted</td>
<td>versus</td>
<td>t</td>
</tr>
<tr>
<td>mph</td>
<td>versus</td>
<td>speed</td>
</tr>
<tr>
<td>maximum</td>
<td>versus</td>
<td>m</td>
</tr>
</tbody>
</table>
The basic syntax for declaring variables is:

typename identifier;
typename identifier = expression;

A variable has the following properties:

• memory location to store the value.
• type of data stored in the memory location.
• name used to refer to the memory location.

Note: You may find the definitive reference for the Java programming language
4.3. Declaration versus Creation

A declaration declares and connects a variable with a type.

```java
int index;
String aString;
```

A creation creates an object.

The following programs have problems:

```java
/**
 * The program will not work.
 * What is the problem?
 * /

class WillNotCompile {
    public static void main(String args[])
    {
        String aString;
        aString.length();
    }
}
```

Source Code: Src/4/WillNotCompile.java

```bash
yps 4 62 javac WillNotCompile.java
WillNotCompile.java:18: Variable aString may not have been initialized.
   aString.length();
1 error
```
1 /**
2  * The program will die.
3  * What is the problem?
4  *
5  * @version $Id$
6  *
7  * @author hp bischof
8  *
9  * Revisions:
10  * $Log$
11  */
12
13 class WillDie
14 {
15     public static void main(String args[])
16     {
17         String aString = new String();
18         aString = null;
19         aString.length();
20     }
21 }
22
Source Code: Src/4/WillDie.java

% javac WillDie.java
% java WillDie
java/lang/NullPointerException
    at WillDie.main(WillDie.java:19)
4.4. String Object/Class

- Strings are constants
- All literals are instances of the String class and exist once in a JVM.
- Their value can not be modified - why?

4.5. The First Program

```java
/**
 * Deal with Strings objects.
 * @version $Id$
 * @author hp bischof
 * Revisions:
 * $Log$
 */
class StringThing {
    public static void main(String args[]) {
        String aString;
        aString = new String("Last Stop Wonderland! ");
        System.out.println( aString.length() );
        System.out.println( aString.toUpperCase() );
        System.out.println( aString.toUpperCase() + "." );
        System.out.println( aString.length() + 1 );
        System.out.println( 1 + aString.length() );
        System.out.println( 1 + aString + 1 );
        System.out.println( aString + ( 1 + 1 ) );
    }
}
```

Result:

20
LAST STOP WONDERLAND
LAST STOP WONDERLAND.

Other Example

```java
/**
 * "abc" versus new String("abc")
 */
class StringL {
```
public static void method(String id, String literal, String aNewString) {
    System.out.println(id + " in method");
    System.out.print("literal= aNewString\n ");
    System.out.println(literal == aNewString);
}

public static void main( String args[] ) {
    String aString = "abc";
    System.out.print("abc == aString\n ");
    System.out.println("abc" == aString);

    String newString = new String("abc");
    System.out.print("abc == new String(abc)\n ");
    System.out.println("abc" == newString);

    method("1", "abc", "abc");
    method("2", "abc", new String("abc") );
    method("3", "abc", "ab" + "c");
    method("4", "abc", "" + "abc");
}

Source Code: Src/4/StringL.java

4.6. From the Bridge Exam

import java.util.*;

public class X_s1 {
    private String info;

    public X_s1 (String info) {
        this.info = info;
    }

    private String info() {
        return info;
    }

    public static void main (String args []) {
        X_s1 one = new X_s1("a");
        X_s1 two = new X_s1("a");

        if ( one.info() == "a") // 1 marked
            System.out.println("1. equal");
        if ( one.info() == two.info() ) // 2 marked
            System.out.println("2. equal");
        if ( one.info().equals(two.info()) ) // 3 marked
            System.out.println("3. equal");
        if ( one.info() == "aa".substring(0,1) ) // 4 marked
            System.out.println("4. equal");
        if ( one.info().equals("aa".substring(0,1) ) ) // 5 marked
            System.out.println("5. equal");
    }
}
4.7. Use of the StringThing Class

```java
/**
 * Play with the String class
 * @version $Id$
 * @author Hpb
 * $Log$
 */

class String_1 {

    public static void main( String args[] ) {
        String aString = "David";
        String bString = "David Bowie";

        if ( " hello".equals("hello") )
            System.out.println("equal");
        if ( "David" == aString )
            System.out.println("David == aString ");
        System.out.println(aString.length());
        System.out.println(aString.charAt(0));

        System.out.println(aString.indexOf("vid"));
        System.out.println(aString.substring(2,3));
        System.out.println(aString.substring( aString.indexOf("a"),
               aString.indexOf("i") ));

        System.out.println(aString.concat("Bowie").length());
        System.out.println(aString.substring(0, aString.length()));
        System.out.println("-->"+help + "<--");
        if ( "David" == help )
            System.out.println("David == help ");
        if ( "David" == aString )
            System.out.println("David == bString ");
    }
}
```

Source Code: Src/4/String_1.java
4.8. Strings and Numbers

```java
class StringAndInteger {
    public static void main(String args[]) {
        System.out.println("Well, 3 + 4 = "+7);
        System.out.println("Well, 3 + 4 = " + 3 + 4);
        System.out.println("Well, 3 + 4 = " + (3 + 4));
    }
}
```

Source Code: Src/4/StringAndInteger.java
class StringAndInteger2 {
    public static void main(String args[]) {
        System.out.println(3 + 7);
        System.out.println(3 + 7 + "abc");
        System.out.println(3 + 7 + "abc" + 1);
        System.out.println(3 + 7 + "abc" + 1 + 2);
        System.out.println("" + 3 + 7 + "abc" + 1 + 2);
        System.out.println("" + (3 + 7) + "abc" + (1 + 2));
    }
}

Source Code: Src/4/StringAndInteger2.java
4.9. More on Strings

```java
class StringUse {
    public static void compare(String aString, String bString) {
        if (aString.equals(bString)) {
            System.out.println("equal");
        } else {
            System.out.println("! equal");
        }
        if (aString == bString) {
            System.out.println("== ");
        } else {
            System.out.println("!= ");
        }
    }
    public static void main(String args[]) {
        String aString = "David";
        String bString = "David";
        compare(aString, bString);
        System.out.println("Using New");
        aString = new String("David");
        bString = new String("David");
        compare(aString, bString);
        System.out.println("Concatenation 1");
        aString = "Da" + "vid";
        bString = "D" + "a" + "vid";
        compare(aString, bString);
        System.out.println("Concatenation 2");
        aString = "Da" + "vid";
        bString = "D" + "a" + "vid";
        compare(aString, bString);
    }
}

Source Code: Src/4/StringUse.java
```
% java StringUse
  equal
  ==
  Using New
equal
! ==
 Concatenation 1
equal
==
 Concatenation 2
equal
==
4.10. Confusion about this

```java
/**
 * Use of this!
 */

class UseOfThis {
    int id;
    UseOfThis(int id) {
        this.id = id;
    }
    private void method_2() {
        System.out.println("method_2: " + this);
    }
    private void method_1() {
        System.out.println("method_1: " + this);
        this.method_2();
        method_2();
    }
    public String toString() {
        return "" + id;
    }
    public static void main(String args[]) {
        UseOfThis aUseOfThis = new UseOfThis(1);
        UseOfThis bUseOfThis = new UseOfThis(2);
        System.out.println(aUseOfThis);
        System.out.println(bUseOfThis);
        aUseOfThis.method_1();
        bUseOfThis.method_1();
    }
}
```

Source Code: Src/4/UseOfThis.java
4.11. Primitive Types and Values

A primitive type is predefined by the Java language and named by a reserved keyword. Please see also Remember:

A variable has the following properties:

• memory location to store the value.
• type of data stored in the memory location.
• name used to refer to the memory location.

Java knows the following types:

<table>
<thead>
<tr>
<th>type</th>
<th>#bits</th>
<th>def. v.</th>
<th>minimum value</th>
<th>maximum value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bits</td>
<td>0</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>char</td>
<td>16 bits</td>
<td>0</td>
<td>0</td>
<td>65535</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>0</td>
<td>-32768</td>
<td>32767</td>
</tr>
<tr>
<td>int</td>
<td>32 bits</td>
<td>0</td>
<td>-2147483648</td>
<td>2147483647</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>0</td>
<td>-9223372036854775808</td>
<td>9223372036854775807</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td>0.0</td>
<td>-3.40282347E38</td>
<td>3.40282347E38</td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>0.0</td>
<td>-1.79E+308</td>
<td>1.79E308</td>
</tr>
</tbody>
</table>

Constants as in ANSI-C, constants consist of a decimal mantissa with a decimal point and an exponent with prefix e or E and optional sign. Many parts can be omitted, but one of a decimal point, an exponent, or a suffix must be present.

Constants are float only with the suffix f or F. With or without the suffix d or D they are double.

With the methods Float.intBitsToFloat() and Double.longBitsToDouble() one can change bitwise representations into floating point values.
Examples:

```java
int index;
int milesPerHour, maximumSpeed;
float pressure, sizeOfme;
double starTrekSpeed;
int picturesTakenSofar = 20;
double probability = 0.789;
```

Conditions can only be expressed with boolean values.

`boolean` has the predefined constants

- `true`
- `false`

The names are not reserved; however, they are only recognized as boolean constants.
4.12. Unicode

Skip
See also

It is necessary for a modern environment to handle, uniformly and comfortably, the textual representation of all the major written languages.

Unicode Standard defines a uniform 16-bit code based on the principle of unification:
two characters are the same if they look the same even though they are from different languages.

This principle, called Han unification, allows the large Japanese, Chinese, and Korean character sets to be packed comfortably into a 16-bit representation.

The UTF encoding of the Unicode Standard is backward compatible with ASCII.
Letting numbers be binary, a rune \( c \) is converted to a multibyte UTF sequence as follows: (See also

1. \( c \) in \([00000000.0bbbbbbb]\) 0bbbbbbb
2. \( c \) in \([00000bbb.bbbbbbb]\) 110bbbbbb, 10bbbbbb
3. \( c \) in \([bbbbbbbb.bbbbbbbb]\) 1110bbbbbb, 10bbbbbb, 10bbbbbb

A byte in the ASCII range 0-127 represents itself in UTF.
Thus UTF is backward compatible with ASCII.
4.13. String to int

```java
/**
 * Converts a String to an int - this is one way out of many
 */

class StringToInt {
  public static void main(String args[])
  {
    int i;
    Integer aInt = new Integer("4");
    i = aInt.intValue();
    i = Integer.parseInt("4");
  }
}

Source Code: Src/4/StringToInt.java

See also:

- Exercises:

4.15. Arithmetic Operators

- the table below shows some of the arithmetic operators that Java provides, in the order of their precedence.
- parentheses can be used to change the order of evaluation
- an arithmetic expression returns (calculates) a value when executed.

<table>
<thead>
<tr>
<th>binary Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td>/</td>
<td>integer division, if both operands are integer; real division otherwise</td>
</tr>
<tr>
<td>%</td>
<td>remainder</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>bit shift left</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>bit shift right</td>
</tr>
<tr>
<td>&gt;&gt;&gt;</td>
<td>unsigned right shift</td>
</tr>
<tr>
<td>&amp;</td>
<td>bitwise and</td>
</tr>
<tr>
<td>^</td>
<td>bitwise xor</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bit operation example:

```java
class BitPrint {
    private void printBytes(String what, int value) {
        System.out.print(what + "\t=\t" + value + "\t=\t");
        for (int index = 31; index >= 0; index --) {
            if (((1 << index) & value) == (1 << index))
                System.out.print("1");
            else
                System.out.print("0");
        }
        System.out.println();
    }

    public static void main(String args []) {
        BitPrint aBitPrint = new BitPrint();
        aBitPrint.printBytes("3\t", 3);
        aBitPrint.printBytes("4\t", 4);
        aBitPrint.printBytes("7\t", 7);
        aBitPrint.printBytes("-3\t", -3);
        aBitPrint.printBytes("-4\t", -4);
        aBitPrint.printBytes("-7\t", -7);
        aBitPrint.printBytes("5\t", 5);
        aBitPrint.printBytes("5 >> 1\t", (5 >> 1));
        aBitPrint.printBytes("-5\t", -5);
        aBitPrint.printBytes("-5 >> 1\t", (-5 >> 1));
        aBitPrint.printBytes("-5 >>> 1\t", (-5 >>> 1));
    }
}
```

Source Code: Src/4/BitPrint.java
Result:

```
javac BitPrint.java && java BitPrint
3  =  3  = 00000000000000000000000000000011
4  =  4  = 00000000000000000000000000000100
7  =  7  = 00000000000000000000000000000111
-3 = -3  = 11111111111111111111111111111101
-4 = -4  = 11111111111111111111111111111100
-7 = -7  = 11111111111111111111111111111101
5  =  5  = 00000000000000000000000000000101
5  >> 1 = 2  = 00000000000000000000000000000010
-5 = -5  = 11111111111111111111111111111101
-5 >> 1 = -3 = 11111111111111111111111111111101
-5 >>> 1 = 2147483645 = 01111111111111111111111111111101
```

4.16. Unary/Binary Operator

<table>
<thead>
<tr>
<th>unary Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>increment by 1</td>
</tr>
<tr>
<td>--</td>
<td>decrement by 1</td>
</tr>
<tr>
<td>!</td>
<td>not</td>
</tr>
</tbody>
</table>

Example:

```java
int left;
int right;
int result;

left  = 4;
right = 3;
result = 4 * 3;
result = 4 + 3;
result = 4 / 3;
result = 4 % 3;

left ++;
result = left++;
result = ++left;

right ++;
result = right++;
result = ++right;
```
4.17. Mixed Mode Arithmetic and Casting

When an expression contains more than one arithmetic type all are converted to the heaviest.

byte → char → short → int → long → float → double

For example, 2 + 3.3 is interpreted as 2.0 + 3.3.
Java is strongly typed. However, the type of the results of evaluating an expression may be changed using casting. To cast, place the target type in parentheses before the operand or expression to be converted.
For example, if we really wanted the results of 2 + 3.3 to be integer, we could use

```java
int index;

index = 2 + (int) 3.3;
index = (int) (2 + 3.3);
```
Example:

```java
1     /**
2     * The program deals with operators.
3     * Comment not included.
4     *
5     * @version $Id$
6     *
7     * @author hp bischof
8     *
9     * Revisions:
10     * $Log$
11     */
12
13 class OpEx
14 {
15     public static void main(String args[]) {
16         char aChar = 'b';
17         byte aByte = 2;
18
19         int intVar_1 = 1;
20         int intVar_2 = 2;
21         int intRes = 3;
22         double doubleVar_1 = 3.8;
23         double doubleVar_2 = 4.8;
24         double doubleRes = doubleVar_1 - doubleVar_2;
25
26         System.out.println("1. "+aChar);    // man ascii decimal set
27         System.out.println("2. "+aByte);
28         System.out.println("3. "+aByte+aChar);
29         System.out.println("4. "+aByte+0);
30         System.out.println("5. "+aChar+0);
31
32         intRes = 5 / 3; System.out.println("6. "+intRes);
33         intRes = 5 % 3; System.out.println("7. "+intRes);
34         // intRes = 5 / doubleVar_2;     // Doesn’t work, why?
35         intRes = (int)(5 / doubleVar_2); System.out.println("8. "+intRes);
36         doubleRes = 5 / doubleVar_2; System.out.println("9. "+doubleRes);
37         doubleRes = 5.0 / doubleVar_2; System.out.println("10. "+doubleRes);
38     }
39 }
```

Source Code: Src/4/OpEx.java

% javac OpEx.java
% java OpEx
1. b
2. 2
3. 2b
4. 20
5. b0
6. 1
7. 2
8. 1
9. 1.0416666666666667
10. 1.0416666666666667
4.18. Assignment Operators

There are 12 assignment operators; all are syntactically right-associative (they group right-to-left). See also

\[ \begin{align*}
= & \quad / \quad \% \quad += \quad -= \\
<<= & \quad >>= \quad >>>= & \quad &= \quad ^\dagger = \quad |= 
\end{align*} \]

\begin{itemize}
  \item \texttt{<<=} \quad \text{bit shift left}
  \item \texttt{>>=} \quad \text{bit shift right}
  \item \texttt{>>>=} \quad \text{unsigned right shift}
  \item \texttt{&=} \quad \text{and}
  \item \texttt{^=} \quad \text{xor}
  \item \texttt{|=} \quad \text{or}
\end{itemize}

The syntax for an assignment is:

```
Assignment:
  LeftHandSide AssignmentOperator AssignmentExpression
```

- \texttt{LeftHandSide} must be a variable
- \texttt{AssignmentOperator} must be one of \texttt{=} \texttt{/=} \texttt{=} \texttt{+=} \texttt{-=} \texttt{<<=} \texttt{>>=} \texttt{>>>=} \texttt{&=} \texttt{^=} \texttt{|=}
- \texttt{AssignmentExpression} must be \texttt{ConditionalExpression} or \texttt{Assignment}

Note: A variable that is declared final cannot be assigned to.
4.19. Playing with the Basic Types

```java
class Ranges {
    /** uses complement operations */
    short x;
    void intRange () {
        System.out.println("int\t" + (~0 >>> 1) + "\t" + (~ (~0 >>> 1)));
        printBytes("(~0 >>> 1)", (~0 >>> 1));
        printBytes("(~ (~0 >>> 1))", (~ (~0 >>> 1)));
    }
    /** maximum and minimum long value */
    static final long maxLong = ~0L >>> 1;
    static final long minLong = ~ (~0L >>> 1);
    private static void printBytes (String what, int value) {
        System.out.print(what + "\t=" + value + "\t=");
        for (int index = 31; index >= 0 ; index --) {
            if (((1< <index ) & value ) == ( 1 << index ) )
                System.out.print("1");
            else
                System.out.print("0");
        }
        System.out.println();
    }
    /** uses casts and literals */
    void shortRange () {
        System.out.println("short\t" + Short.MIN_VALUE + "\t" + Short.MAX_VALUE);
        System.out.println("short\t" + (short)077777 + "\t" + (short)0x8000);
    }
    /** shifts ones until no further changes occur */
    void byteRange () {
        ...
    }
}
```
byte i, j = 1;

do {
    i = j; j = (byte)(i << 1 | 1);
} while (j > i);
System.out.print("byte\t" + i);

do {
    i = j; j = (byte)(i << 1);
} while (j < i);
System.out.println("\t" + i);

public static void main (String args []) {
    Ranges aRange = new Ranges();
    aRange.byteRange();
    aRange.shortRange();
    aRange.intRange();
    aRange.longRange();
}

Source Code: Src/4/Ranges.java
Result:

<table>
<thead>
<tr>
<th>Type</th>
<th>Min Value</th>
<th>Max Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>127</td>
<td>-128</td>
</tr>
<tr>
<td>short</td>
<td>32767</td>
<td>-32768</td>
</tr>
<tr>
<td>int</td>
<td>2147483647</td>
<td>-2147483648</td>
</tr>
<tr>
<td>long</td>
<td>9223372036854775807</td>
<td>-9223372036854775808</td>
</tr>
</tbody>
</table>

**intRange()**

produces a maximal bit pattern by complementing 0 and shifting without propagating the sign, and a minimal bit pattern by complementing once more.

**+** concatenates strings; integer operands are converted to short strings. java/gtexit has facilities for explicit formatting.

**static**
defines `maxLong` and `minLong` as class variables.

**final** permits exactly one assignment. Initialization is done with a long literal and thus long arithmetic.

**shortRange()**

uses int literals which must be explicitly cast to short. The following expression produces a minimal value in C but not in Java (why?):

```
(short) ~ ((unsigned short) ~0 >> 1)
```

**byteRange()**

uses do-while loops. byte can be initialized with int, but for assignment an explicit cast is required.
4.20. Flow Control

4.21. Conditions

• Conditions can only be expressed with boolean values; unlike C, an implicit comparison to zero is not acceptable.

• Boolean has the predefined constants true and false. The names are not reserved; however, they are only recognized as boolean constants.

4.22. Relational Operators

• Simple boolean expressions consist of comparing things using relational operators. There two types of relational operators: equality and comparison.

Equality operators are defined for all objects and primitive types.

==  equal
!=  not equal

All comparisons produce true or false.
4.23. Logical Operators

These operators take boolean arguments and return boolean results. They are used to construct complex boolean expressions from simple ones consisting of boolean values and relational expressions.

- `&` bitwise and
- `&&` conditional and (short circuits)
- `|` bitwise or
- `||` conditional or (short circuits)
- `^` bitwise xor
- `!` not (unary operator)/boolean complement

- `x && y` y will be evaluated only if x is true
- `x || y` y will be evaluated only if x is false
class ShortCut
{
  private boolean testIt(double n) {
      return n != 0.0;
  }
  private double oneDn(double n) {
      return 1.0 / n;
  }
  public static void main(String args[])
  {
    double n;
    ShortCut aS = new ShortCut();
    n = 0.5;
    if (aS.testIt(n) && (aS.oneDn(n) > 1))
      System.out.println("1: 1 / n " + 1 / n);
    n = 0;
    if (aS.testIt(n) && (aS.oneDn(n) > 1))
      System.out.println("2: 1 / n " + 1 / n);
    System.out.println("3: 1.0 / 0 = " + 1.0 / 0);
    if ((n==0) || (n==1))
      System.out.println("4: ( n == 0 ) || ( n == 1 )");
    System.out.println("5. true || false && true: " + (true || false && true));
    if (4==(n=4))
      System.out.println("6. 4 == ( n = 4 )");
  }
}

Source Code: Src/4/ShortCut.java

Result:
% javac ShortCut.java && java ShortCut
1: 1 / n 2.0
3: 1.0 / 0 = Infinity
4: ( n == 0 ) || ( n == 1 )
5. true || false && true: true
6. 4 == ( n = 4 )
The precedence for the arithmetic, relational, boolean operators, and assignment from highest to lowest is:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Symbol</th>
<th>Precedence</th>
<th>Association</th>
</tr>
</thead>
<tbody>
<tr>
<td>grouping</td>
<td>(</td>
<td>1</td>
<td>left to right</td>
</tr>
<tr>
<td>unary</td>
<td>+, -</td>
<td>2</td>
<td>right to left</td>
</tr>
<tr>
<td>multiplication</td>
<td>*</td>
<td>3</td>
<td>left to right</td>
</tr>
<tr>
<td>division</td>
<td>/</td>
<td>3</td>
<td>left to right</td>
</tr>
<tr>
<td>remainder</td>
<td>%</td>
<td>3</td>
<td>left to right</td>
</tr>
<tr>
<td>addition</td>
<td>+</td>
<td>4</td>
<td>left to right</td>
</tr>
<tr>
<td>subtraction</td>
<td>-</td>
<td>4</td>
<td>left to right</td>
</tr>
<tr>
<td>less than</td>
<td>&lt;</td>
<td>5</td>
<td>left to right</td>
</tr>
<tr>
<td>less than or equal</td>
<td>&lt;=</td>
<td>5</td>
<td>left to right</td>
</tr>
<tr>
<td>greater than</td>
<td>&gt;</td>
<td>5</td>
<td>left to right</td>
</tr>
<tr>
<td>greater than or equal</td>
<td>&gt;=</td>
<td>5</td>
<td>left to right</td>
</tr>
<tr>
<td>equal</td>
<td>==</td>
<td>6</td>
<td>left to right</td>
</tr>
<tr>
<td>not equal</td>
<td>!=</td>
<td>6</td>
<td>left to right</td>
</tr>
<tr>
<td>bit and</td>
<td>&amp;</td>
<td>7</td>
<td>left to right</td>
</tr>
<tr>
<td>xor</td>
<td>^</td>
<td>8</td>
<td>left to right</td>
</tr>
<tr>
<td>bit or</td>
<td></td>
<td>9</td>
<td>left to right</td>
</tr>
<tr>
<td>conditional and</td>
<td>&amp;&amp;</td>
<td>10</td>
<td>left to right</td>
</tr>
<tr>
<td>conditional or</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>assignment</td>
<td>=, +=, *= ...</td>
<td>12</td>
<td>N.A.</td>
</tr>
</tbody>
</table>
4.24. if Statement

The if statement allows conditional execution of a statement or a conditional choice of two statements, executing one or the other but not both.

IfThenStatement:

if ( Expression )
  Statement

Example:

x = 3;
y = 4;
if ( x > y )
  z = x;

IfThenElseStatement:

if ( Expression )
  Statement
else
  Statement

Example:

x = 3;
y = 4;
if ( x > y )
  z = x;
else
  z = y;

    class If
    {
      public static void main(String args[]) {
        int index = 2;
        System.out.println("1. " + index);
        if ( ++index == 2 )
          System.out.println("2. " + index);
        else if ( index++ == 3 )
          System.out.println("2. " + index);
        else
          System.out.println("2. " + index);
      }
    }

Source Code: Src/4/If.java

% java If
1. 2
2. 4
4.25. Find the Maximum of two Numbers I

/ **
  * Find the maximum of two numbers.
  * @version $Id$
  * @author Hpb
  * Revisions:
  * $Log$
  */

class Maximum_2 {
  public static double maximum(double _first, double _second ) {
    double max;
    // find maximum
    if ( _first > _second )
      max = _first;
    else
      max = _second;
    return max;
  }

  private double minimum(double _first, double _second ) {
    double minimum;
    // find minimum
    if ( _first < _second )
      minimum = _first;
    else
      minimum = _second;
    return minimum;
  }

  public static void main( String args[] ) {
    Maximum_2 aMaximum_2 = new Maximum_2();
    double firstN = 42.0;
    double secondN = 666.0;
    System.out.println("Maximum(" + firstN + ", " + secondN + ") = " +
                        aMaximum_2.maximum( firstN, secondN) );
    System.out.println("Minimum(" + firstN + ", " + secondN + ") = " +
                        aMaximum_2.minimum( firstN, secondN) );
  }
}

Source Code: Src/4/Maximum_2.java

As a Nassi Shneidermann diagram:

public static double maximum(double _first, double _second )
if if ( _first > _second ) then Yes : return _first else No
    return _second }
public static double minimum(double _first, double _second )
if if ( _first < _second ) then Yes : return _first else No
return _second }
4.26. The conditional operator

?:

uses the boolean value of one expression to decide which of two other expressions should be evaluated.
The conditional operator is syntactically right-associative (it groups right-to-left), so that

\[ a \ ? \ b \ : \ c \ ? \ d \ : \ e \ ? \ f \ : \ g \]

means the same as

\[ a \ ? \ b \ : (c \ ? \ d \ : (e \ ? \ f \ : g)) \].

The conditional operator may be used to choose between second and third operands of numeric type, or
second and third operands of type boolean, or second and third operands that are each of either reference
type or the null type. All other cases result in a compile-time error.
See also:

```java
class QuestionM {
    public static void main(String args[]) {
        int value = 2 > 3 ? 2 : 3;
        String aString = "2 > 3 ? 2 : 3";
        System.out.println(aString + " = " + value);

        value = (1 > 2 ? 3 : (4 < 5 ? 6 : 7));
        aString = "(1 > 2 ? 3 : (4 < 5 ? 6 : 7))";
        System.out.println(aString + " = " + value);

        value = 1 > 2 ? 3 : 4 > 5 ? 8 : 7;
        aString = "1 > 2 ? 3 : 4 > 5 ? 8 : 7";
        System.out.println(aString + " = " + value);
    }
}
```

Source Code: Src/4/QuestionM.java

% java QuestionM
2 > 3 ? 2 : 3 = 3
( 1 > 2 ? 3 : ( 4 < 5 ? 6 : 7 )) = 6
1 > 2 ? 3 : 4 > 5 ? 8 : 7 = 7
Example:

```java
class Maximum_3 {
    private double maximum(double _first, double _second) {
        return _first > _second ? _first : _second;
    }

    public static double minimum(double _first, double _second) {
        return _first < _second ? _first : _second;
    }

    public static void main(String args[]) {
        Maximum_3 aMax = new Maximum_3();
        double firstN = 42.0;
        double secondN = 7.0;

        System.out.println("Maximum(" + firstN + ", " + secondN + ") = " + aMax.maximum(firstN, secondN));
        System.out.println("Minimum(" + firstN + ", " + secondN + ") = " + aMax.minimum(firstN, secondN));
    }
}
```

Source Code: Src/4/Maximum_3.java
4.27. while Statement

See also

The while statement executes an Expression and a Statement repeatedly until the value of the Expression is false.

WhileStatement:

\[
\text{while ( Expression )} \\
\text{Statement}
\]

Example:

\[
x = 1; \\
\text{while ( x < 10 )}{ \\
\text{\quad print x} \\
\text{\quad x += 2;} \\
}\]

```java
class While {
    public static void main(String args[]) {
        int index = 1;
        while ( index > 0 ? ( index == 7 ) : (index == 8 ) ) {
            System.out.println("index = " + index );
        }
        System.out.println("index =" + index );
    }
}

Source Code: Src/4/While.java

% java While
index = 1
```

```java
class While_1 {
    public static void main(String args[]) {
        int index = 1;
        while ( ++index++ < 4 ) {
            System.out.println("index = " + index );
        }
        System.out.println("index = " + index );
    }
}

Source Code: Src/4/While_1.java
```
class While_2 {
    public static void main(String args[]) {
        int index = 1;
        while ( ++index < 4 ) {
            System.out.println("index = " + index);
        }
        System.out.println("index = " + index);
    }
}

Source Code: Src/4/While_2.java

% java While_2
index = 2
index = 3
index = 4
The Expression must have type boolean, or a compile-time error occurs.

A while statement is executed by first evaluating the Expression. If evaluation of the Expression completes abruptly for some reason, the while statement completes abruptly for the same reason. Otherwise, execution continues by making a choice based on the resulting value:

If the value is true, then the contained Statement is executed. Then there is a choice:

If execution of the Statement completes normally, then the entire while statement is executed again, beginning by re-evaluating the Expression.

If the value of the Expression is false, no further action is taken and the while statement completes normally.

If the value of the Expression is false the first time it is evaluated, then the Statement is not executed.
4.28. Calculate Sqrt(2) without the MathClass

Algorithm:
\[
\text{n1} = 1.0; \quad \text{n2} = 2.0; \quad \text{index} = 1 \quad \text{do} \quad \text{while} \quad \text{n2} \times \text{n2} - \text{n1} \times \text{n1} > 0.0001: \quad \text{x} = \text{n1} + (\text{n2} - \text{n1}) \times 0.5 \\
\text{if} \quad \text{x} \times \text{x} > 2 \quad \text{then} \quad \text{YES:} \quad \text{n2} = \text{x} \quad \text{else} \quad \text{NO:} \quad \text{n1} = \text{x} \quad \text{end if}
\]
class Sqrt {
  private double calculateSqrt_2() { 
    double n1 = 1.0;
    double n2 = 2.0;
    while ( (n2 * n2 - n1 * n1) > 0.0001) { 
      double x = (n2 + n1) * 0.5;
      if (x * x > 2.0) 
        n2 = x;
      else
        n1 = x;
    }
    return n1;
  }
  public static void main( String args[] ) { 
    System.out.println("sqrt(2) = " +
        new Sqrt().calculateSqrt_2() + " +- 0.0001 ");
  }
}

Source Code: Src/4/Sqrt.java
4.29. Continue

- continue statement may occur only in a while, do, or for statement;
- continue statement with no label attempts to transfer control to the innermost enclosing while, do, or for statement;
- this statement, which is called the continue target, then immediately ends the current iteration and begins a new one.
- If no while, do, or for statement encloses the continue statement, a compile-time error occurs.

Example 1:

```java
class Continue_1 {
    public static void main( String args[] ) {
        int n = 0;
        while ( n < 6 ) {
            System.out.println("1. n == "+n);
            while (n<4) {
                n++;
                System.out.println(" 2. n == "+n);
                if (n>2)
                    continue label1;
                System.out.println(" 3. n == "+n+"--------");
            }
            n++;
        }
    }
}
```

Source Code: Src/4/Continue_1.java

% java Continue_1
1. n == 0
   2. n == 1
   3. n == 1--------
      2. n == 2
      3. n == 2--------
      2. n == 3
1. n == 3
   2. n == 4
1. n == 4
1. n == 5

4.30. Break

- A break statement transfers control out of an enclosing iteration or switch statement.
- Labeled breaks are used to jump out of nested loops
• break statement with no label attempts to transfer control to the innermost enclosing switch, while, do, or for statement;
• If no switch, while, do, or for statement encloses the break statement, a compile-time error occurs.

Example 1:

```java
class Break_1 {
    public static void main( String args[] ) {
        int n = 0;
        here: {
            while ( true ) {
                System.out.println("1. n == " + n);
                while ( n < 100 ) { // while ( true ) --> which problem
                    n++;
                    System.out.println("2. n == " + n);
                    if ( n > 2 )
                        break here;
                }
                System.out.println("3. n == " + n);
            }
        }
    }
    Source Code: Src/4/Break_1.java
}
```

Example 2:

```java
class Break_2 {
    public static void main( String args[] ) {
        int n = 0;
        System.out.println("start");
        while ( n < 100 ) {
            if ( n > 4 )
                System.exit(1);
            while ( n < 100 ) { // while ( true ) --> which problem
                n++;
                System.out.println("inner while here n == " + n);
                if ( n > 2 )
                    break;
            }
            System.out.println("outer while here n == " + n);
        }
        System.out.println("after here ");
    }
```
4.31. Return

A return statement returns control to the invoker of a method or constructor.

```java
/* How can we set the exit code? */
class Return {
    public static void main( String args[] ) {
        int x = 0;
        return x;
    }
}
```

Source Code: Src/4/Return.java

```java
/* How can we set the exit code? */
class Return_1 {
    public static int method() {
        System.exit(2);
        return 0;
    }
    public static void main( String args[] ) {
        method();
        System.out.println("xxx");
    }
}
```

Source Code: Src/4/Return_1.java
4.32. **Return vs. Continue vs. Break**

- What are the differences?

4.33. **Abrupt Completion**

Abrupt completion of the contained Statement is handled in the following manner:

An abrupt completion always has an associated reason, which is one of the following: (from

- A break with no label
- A break with a given label
- A continue with no label
- A continue with a given label
- A return with no value
- A return with a given value
- A throw with a given value, including exceptions thrown by the Java virtual machine
```java
class Break {
    public static void main( String args[] ) {
        int n = 0;

        here: {
            while ( true ) {
                System.out.println("a: outer while here n == " + n);
                if ( n > 4 )
                    System.exit(1);

                while ( true ) {
                    System.out.println(" inner while here n == " + n);
                    if ( ++n == 0 )
                        System.out.println("n == 0");
                    else if ( n++ == 1 ) {
                        System.out.println(" n = 1");
                        System.out.println(" break");
                        break;
                    } else if ( n++ == 2 )
                        System.out.println(" n == 2");
                    else
                        System.out.println(" n == 3");
                    System.out.println(" executing break here");
                    System.out.println(" n is " + n);
                    break here;
                }
                System.out.println("b: outer while here n == " + n);
            }
            // unreachable statement ...System.out.println("here ");
        }

        Source Code: Src/4/Break.java
    }
```
4.34. do Statement

See also

The do statement executes a Statement and an Expression repeatedly until the value of the Expression is false.

do Statement while ( Expression ) ;
4.35. for Statement

See also

The for statement executes some initialization code, then executes an Expression, a Statement, and some update code repeatedly until the value of the Expression is false.

ForStatement:

```java
for ( ForInit; Expression; ForUpdate)
  Statement
```

Example:

```java
class For_1 {
  public static void main( String args[] ) {
    int index = 0;
    for ( index = 0 ; index < 1; index ++ ) {
      System.out.println("1. index = " + index );
    }
    System.out.println("2. index = " + index );
  }
}
Source Code: Src/4/For_1.java
```

```java
class For_2 {
  public static void main( String args[] ) {
    int index = 0;
    for ( index = 0 ; index < 1; index ++ ) {
      index = -1;
      System.out.println("1. index = " + index );
      break;
    }
    System.out.println("2. index = " + index );
  }
}
Source Code: Src/4/For_2.java
```

```java
class For_3 {
  public static void main( String args[] ) {
    for ( int index = 0 ; index < 1; index ++ ) {
      System.out.println("1. index = " + index );
      break;
    }
    System.out.println("2. index = " + index );
  }
}
Source Code: Src/4/For_3.java
```
4.36. Find all Prime Numbers in [ 2 ... 100 ]

```python
isPrime(n): do for index = 2 to n - 1 : if ( index % n == 0 ) : return false } return true
findAllPrimeN(): do for index = 1 to 100 : if ( isPrime(n) ) : print index }
```
class Prime_1 {

    private boolean isPrime(int n) {
        for (int index = 2; index < n; index++) {
            if (n % index == 0)
                return false;
        }
        return true;
    }

    public static void main(String args[]) {
        Prime_1 aPrime = new Prime_1();
        for (int index = 2; index <= 10; index++)
            if (aPrime.isPrime(index))
                System.out.println(index + " ");
    }

    Source Code: Src/4/Prime_1.java
4.37. Switch Statement

See also

The switch statement transfers control to one of several statements depending on the value of an expression. The type of the switch expression can be

- char
- byte
- short
- int
- strings
- enum type

Switch Statement:

```java
switch ( Expression ) {
    case ConstantExpression_1 : action_1;
    case ConstantExpression_2 : action_2;
    ...
    default: action_d
```
Example:

```java
class Switch {

  static void method(int k) {
    switch (k) {
      case 1: System.out.println("with break: 1 ");
          break;
      case 2: System.out.println("with break: 2 ");
          break;
      default: System.out.println("with break: default");
    }
  }

  static void methodWithoutDefault(int k) {
    switch (k) {
      case 1: System.out.println(" without break: 1 ");
          break;
      case 2: System.out.println(" without break: 2 ");
          break;
    }
  }

  public static void main(String[] args) {
    new Switch().method(3);
    new Switch().methodWithOutDefault(2);
    new Switch().methodWithOutDefault(3);
  }
}
```

Source Code: Src/4/Switch.java

### 4.38. Partial Lowercase → Uppercase

```java
/**
 * Test of the switch statement.
 * @version $Id$
 * @author hpb
 * Revisions:
 * $Log$
 */

class Switch_1 {
  private String itWasA(char c) {
    switch( c ) {
      case 'a': return("A"); // break??
      case 'b': return("B"); // break??
    }
  }
```

```
18     case 'c':    return("C");   // break??
19     case 100:   return("D");   // break??
20     case 101:   return("E");   // break??
21     default:    return("no clue, but not an [a-e]");
22         // What happens if
23         // we delete this line?
24     }
25 }
26
27     public static void main( String args[] ) {
28         char theChar;
29
30         theChar = 'd';
31         System.out.println("You typed in an '" +
32                 new Switch_1().itWasA(theChar) + "'");
33         System.exit(0); // beware of ...
34     }
35 }
36

Source Code: Src/4/Switch_1.java

Characters can be safely converted to integers (Unicode), but should be avoided at all times.

4.39. Questions

- Which variable names are valid:

```java
1    class X_1
2    {
3        public static void main(String args[])
4        {
5            int aInt;
6            int countUpTo5;
7            int 5IsANiceNumber;
8            int ooo";
9            int notToMany:areAllowed;
10        }
11    }
12
Source Code: Src/4/X_1.java
```

- What is the output of the following program:

```java
1    class X_2
2    {
3        public static void main(String args[])
4        {
5            System.out.println("I like to play " + 6 + 2");
6            System.out.println("I like to play " + 6 * 2);
7            System.out.println("I like to play " + ( 6 + 2 ));
```
Will the following program compile?

```java
class X_3 {
    public static void main(String args[]) {
        int i = 0;
        i += 63;
        System.out.println("1. " + ( i++ >> 2 ) );
        System.out.println("2. " + ( 1 > 2 ? 3 : 6 ));
    }
}
```

What is the output of the following program:

```java
class X_4 {
    public static void main(String args[]) {
        int i = 0;
        i += 63;
        System.out.println("1. " + ( i++ >> 2 ) );
        System.out.println("2. " + ( 1 > 2 ? 3 : 6 ));
        System.out.println("4. " + (1 > 2 ? 3 : (4 < 5 ? 6 : (9 < 10 ? 7 : 8))));
    }
}
```
```java
class X_5 {
    public static void main( String args[] ) {
        int n = 0;
        while ( true ) {
            System.out.println("xx");
            if ( n++ == 0 ) {
                System.out.println("n == 0");
            } else if ( n++ == 1 ) {
                System.out.println("n == 1");
            } else if ( n-- == 2 )
                System.out.println("n == 2");
        }
    }
}
```

Source Code: Src/4/X_5.java
5. Scanner: Overview

- Introduced to satisfy faculty, students, and anyone who wants to write a quick-and-dirty Java program that uses console I/O.
- Works like StreamTokenizer
- Implements Iterator<String>
- See here:
- Perl-like pattern matching available
- See here:

5.1. Scanner: Constructors

- Scanner( File source )
- Scanner( InputStream source )
- Scanner( String source )
- System.in is an InputStream
- There are also constructors to work with:
  alternate character sets; input objects from the java.nio library.

5.2. Scanner: Reading Methods

- String next()
- String next( Pattern pattern )
- boolean nextBoolean()
- double nextDouble()
- int nextInt()
- int nextInt( int radix )
- String nextLine()

5.3. Scanner: Testing Methods

- boolean hasNext()
- boolean hasNext( Pattern ptrn )
- boolean hasNextBoolean()
- boolean hasNextDouble()
- boolean hasNextInt()
- boolean hasNextInt( int radix )
- boolean hasNextLine()

5.4. Scanner: Example 1

```java
import java.util.Scanner;

public class Scanner1 {
    public static void main( String[] args ) {
        Scanner sc = new Scanner( System.in);
        System.out.printf("> ");
        while ( sc.hasNext() ) {
```
String line = sc.nextLine();
System.out.printf("-%s-%n", line);
System.out.printf("> ");
}
sc.close();
}

Source Code: Src/6_jdk15/Scanner1.java

% java Scanner1
> 1 2 3 4 hello
-1 2 3 4 hello-
> ups
-ups-
> # "D here ....

5.5. Scanner: Example 2

import java.util.Scanner;

public class Scanner2 {
    public static void main( String[] args ) {
        Scanner sc = new Scanner( System.in);
        System.out.printf("> ");
        while ( sc.hasNext() ) {
            Integer aInteger = sc.nextInt();
            System.out.printf("-%d-%n", aInteger);
            System.out.printf("> ");
        }
        sc.close();
    }
}

Source Code: Src/6_jdk15/Scanner2.java

% java Scanner2
> 1 2 3 4 1.0
-1-
> -2-
> -3-
> -4-
> Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:819)
    at java.util.Scanner.next(Scanner.java:1431)
    at java.util.Scanner.nextInt(Scanner.java:2040)
    at java.util.Scanner.nextInt(Scanner.java:2000)
    at Scanner2.main(Scanner2.java:9)
Reading from a file

5.6. Scanner: Example 3

```java
import java.util.Scanner;

public class Scanner3 {
    public static void main( String[] args ) {
        Scanner sc = new Scanner("1blobblob2blob3").useDelimiter("blob");
        System.out.printf("> ");
        while ( sc.hasNext() ) {
            String line = sc.next();
            System.out.printf("-%s-%n", line);
            System.out.printf("> ");
        }
        sc.close();
    }
}

Source Code: Src/6_jdk15/Scanner3.java

% java Scanner3
> -1-
> --
> -2-
> -3-
```

5.7. Scanner: Example 4

```java
/*
 * example is from: http://www.cs.rit.edu/~hpb/Jdk5/api/java/util/Scanner.html
 */
import java.util.Scanner;
import java.util.regex.MatchResult;

public class Scanner4 {
    public static void printIt(String input) {
        Scanner s = new Scanner(input);
        s.findInLine("(\d+) fish (\d+) fish (\w+) fish (\w+)");
        MatchResult result = s.match();
        for (int i=1; i<=result.groupCount(); i++) {
            System.out.println(i + " :" + result.group(i));
        }
    }

    public static void main( String[] args ) {
        String input = "1 fish 2 fish red fish blue fish";
```
21     printIt(input);
22
23     }
24     }
25
26
Source Code: Src/6_jdk15/Scanner4.java

% java Scanner4
1: 1
2: 2
3: red
4: blue
6. Class Relationships

See also:

- Class declarations define new reference types and describe how they are implemented.
- Constructors are similar to methods, but cannot be invoked directly by a method call; they are used to initialize new class instances. Like methods, they may be overloaded.
- Static initializers are blocks of executable code that may be used to help initialize a class when it is first loaded.
- The body of a class declares members, static initializers, and constructors.
- The scope of the name of a member is the entire declaration of the class to which the member belongs. Field, method, and constructor declarations may include the access modifiers public, protected, or private. The members of a class include both declared and inherited members.
- Newly declared fields can hide fields declared in a superclass or super interface. Newly declared methods can hide, implement, or override methods declared in a superclass.
- Visibility modifier: public/private/protected
- Return type: void/primitive type/reference to a object
- Class methods/Class variables are declared with static.
- Static declaration inside a method change the lifetime of a variable.
6.1. Example

```java
public class X {

    static int cI = 0;
    int oI = 1;

    public X() {
    }

    static public void cm(int i){
        cI = i;
        System.out.println("cm: " + cI);
    }

    public void om(int i){
        cI = i;
        oI = i;
        System.out.println("om: " + this);
        System.out.println(" cI " + cI);
        System.out.println(" oI " + oI);
    }

    public static void main(String args[]) {
        X aX = new X();
        X aaX = new X();

        aX.cm(1);
        aaX.cm(2);

        aX.om(3);
        aaX.om(4);
    }
}
```

Source Code: Src/5/X.java

```
% java X
cm: 1
om: X@1ad086a
cI 3
  oI 3
om: X@10385c1
cI 4
  oI 4
```

6.2. Class Details

6.3. Static in Classes/Methods Lifetime

- Class Variables: If a variable is declared static, there exists exactly one incarnation of the field,
- Static Methods: A method that is declared static is called a class method. A class method is always invoked without reference to a particular object.
• Non Static Methods: A method that is not declared static is called an instance method, and sometimes called a non-static method. An instance method is always invoked with respect to an object, which becomes the current object to which the keywords this and super refer during execution of the method body.

• Variables can be declared:
  — static: class variable
  — final: can be assigned once or zero time
  — transient: not stored or saved via the standard serialization process
  — volatile: A variable may be modified by multiple threads. This gives a hint to the compiler to fetch the value each time, rather store a locale copy. This also prohibits same optimization procedures.

• See also:

```java
public class Overview {
    int instanceVariable;
    static int classVariable;
    final int finalVariable; // static?
    volatile int volatileVariable;
    transient int transientVariable;

    public Overview() {
        finalVariable = 42;
    }

    public Overview(int aLocalVariable) {
        finalVariable = 43;
    }

    void instanceMethod() {
        finalVariable = 43;
        instanceVariable = 22;
        classVariable = 33;
    }

    static void classMethod() {
        classVariable = 3;
    }

    public static void main(String args[]) {
        Overview aOverview = new Overview();
        Overview bOverview = new Overview();
        Overview cOverview = new Overview(1);
        aOverview.instanceMethod();
        instanceMethod();
        bOverview.classMethod();
        // values of aOverview.? bOverview.? // aOverview.finalVariable??
    }
}
```

Source Code: Src/5/Overview.java
6.4. Parameter Passing

- The formal parameters of a method, if any, are specified by a list of comma-separated parameter specifiers.

- Each parameter specifier consists of a type and an identifier (optionally followed by brackets) that specifies the name of the parameter.

- If a method has no parameters, only an empty pair of parentheses appears in the method's declaration.

- If two formal parameters are declared to have the same name (that is, their declarations mention the same identifier), then a compile-time error occurs.

- When the method is invoked, the values of the actual argument expressions initialize newly created parameter variables, each of the declared type, before execution of the body of the method.

- The scope of formal parameter names is the entire body of the method. These parameter names may not be redeclared as local variables or exception parameters within the method; that is, hiding the name of a parameter is not permitted.

- call by value
6.5. Example I

```java
public class ExampleClass {
    int aLocalVariable = 3;

    public ExampleClass() {
        aLocalVariable = 2;
    }

    public ExampleClass(int aLocalVariable) {
        this.aLocalVariable = aLocalVariable;
        aLocalVariable = 6;
    }

    public static void main(String args[]) {
        ExampleClass aExampleClass = new ExampleClass();
        aExampleClass = new ExampleClass(3);
        System.out.println("the value is: "+ aExampleClass.aLocalVariable);
        System.out.println(aLocalVariable);
    }
}
```

Source Code: Src/5/ExampleClass.java

Questions:
- How does the JVM find the main method, when you execute `java ExampleClass`?
- Describe the execution order of the constructors.
- Which variables will be modified when?

6.6. Example II

```java
import java.util.Vector;

public class Args {
    String aString;
    Vector aVector = new Vector();

    public void testString(String arg) {
        arg = "b";
    }

    public void testVector(Vector arg) {
        arg.add("b");
    }

    public void test1() {
        aString = "a";
        System.out.println("1. "+ aString);
        testString(aString);
        System.out.println("2. "+ aString);
    }

    public void test2() {
```
23     aVector.add("a");
24     System.out.println("3. " + aVector);
25     testVector(aVector);
26     System.out.println("4. " + aVector);
27 }
28 public static void main(String args[]) {
29     new Args().test1();
30     new Args().test2();
31 }
32 }

Source Code: Ssrc/5/Args.java

Questions:
• How does the JVM find the the variable aStaticLocalVariable?
• Which variables will be modified when?
6.7. Example III

```java
public class ExampleClassIII {
    String aString = null;

    public void method(String a) {
        a = new String("set in method");
        System.out.println("2. method:a:" + a);
    }

    public void test() {
        String aString = new String("set in test");
        System.out.println("1. test:aString:" + aString);
        method(aString);
        System.out.println("3. test:aString:" + aString);
    }

    public static void main(String args[]) {
        new ExampleClassIII().test();
    }
}
```

Source Code: Src/5/ExampleClassIII.java

6.8. Example IV

```java
public class XX {
    int instanceV = 1;

    static XX bbbbb;
    static XX aaaaaa;

    public XX() {
    }

    public void method(int i){
        instanceV = i;
    }

    public String toString() {
        return "instanceV = " + instanceV;
    }

    public void m2(int i){
        aaaaaa.method(-9);
        method(12);
        System.out.println("-\-------------------------------------------------");
        System.out.println("print itself : " + this);
        System.out.println("print aaaaaa: " + aaaaaa);
        System.out.println("-\-------------------------------------------------");
    }
}
```
27 public static void main(String args[] )
28 {
29     bbbbbb = new XX();
30     aaaaaa = new XX();
31     bbbbbb.m2(3);
32     aaaaaa.m2(24);
33 }
34 }

Source Code: Src/5/XX.java

• does it compile?
• Output?

-----------------------------------------
print itself : instanceV = 12
print aaaaaa: instanceV = -9
=================================================================
print itself : instanceV = 12
print aaaaaa: instanceV = 12
=================================================================

1
2 public class ClassXX {
3     static int instanceV = 1;
4     static ClassXX bbbbbb;
5     static ClassXX aaaaaa;
6
7     public ClassXX() {
8     }
9     public void method(int i){
10         instanceV = i;
11     }
12     }
13
14     public String toString() {
15         return "instanceV = " + instanceV;
16     }
17     }
18
19     public void m2(int i){
20         aaaaaa.method(-9);
21         method(12);
22         System.out.println("---------------");
23         System.out.println("print itself : " + this);
24         System.out.println("print aaaaaa: " + aaaaaa);
25         System.out.println("----------------------");
26     }
27     public static void main(String args[] )
28     {
29         bbbbbb = new ClassXX();
30         aaaaaa = new ClassXX();
31         bbbbbb.m2(3);
6.9. Example VI

```java
public class XXX {
    int oI = 1;

    XXX aXXX = new XXX();

    public XXX() {
    }

    public XXX(XXX aXXX) {
        this.aXXX = aXXX;
    }

    public void m1(int i){
        oI = i;
    }

    public void m2(int i){
        oI = i;
        m1(12);
        aXXX.m1(24);
        System.out.println("om: " + this);
        System.out.println("oI " + oI);
        System.out.println("aXXX.om: " + aXXX);
        System.out.println("aXXX.oI " + aXXX.oI);
    }

    public static void main(String args[] ) {
        XXX aaXXX = new XXX(new XXX());
        aaXXX.m2(3);
    }
}
```

Source Code: Src/5/XXX.java

- does it compile? ja. Draw the memory pic during execution
- Does it execute? Ja, aber es wird sterben, weil im C. ein wort fehlt Output?

6.10. A Point Class

- An example without any comment:

  1. Use of a *Point* Class:
public class TestPoint {
    private static Point aPoint;

    /**
     * The main program.
     *
     * @param args command line arguments (ignored)
     */
    public static void main(String args[]) {
        System.out.println("Point.soManyPoints = " + Point.soManyPoints() );
        System.out.println("x = " + aPoint.getX() );
        System.out.println("y = " + aPoint.getY() );

        aPoint = new Point();
        aPoint.initPoint(4, 5);
        System.out.println("x = " + aPoint.getX() );
        System.out.println("y = " + aPoint.getY() );

        aPoint.move(6, 7);
        System.out.println("x = " + aPoint.getX() );
        System.out.println("y = " + aPoint.getY() );

        System.out.println("nPoints = " + aPoint.getNPoints() );
        System.out.println("aPoint.soManyPoints = " + aPoint.soManyPoints() );
    }
    
    Source Code: Src/5/TestPoint.java
II. The \textbf{Point} Class:

```java
/**
 * This class implements a point in a two dimensional area.
 * All method print when they are called.
 */

public class Point {
    static int nPoints; // so many points were created.
    private int x; // x coordinate of the point
    private int y; // y coordinate of the point

    /**
     * Default Constructor.
     * Increases the counter nPoints by 1.
     *
     * @return Point aPoint object
     */
    public Point(){
        super();
        System.out.println(" in Point() constructor");
        nPoints ++;
    }

    /**
     * Constructor.
     * initialize x and y values of a point
     *
     * @param x x coordinate
     * @param y y coordinate
     *
     * @return Point aPoint object
     */
    public Point(int x, int y){
        super();
        int i ++;
        this.x = x;
        this.y = y;
        System.out.println(" in Point(int, int) constructor");
    }

    /**
     * So many points have been created.
     */
    
```
* So many points have been created */
public static int soManyPoints(){
    return nPoints;
}

/**
 * initializes x and y of a point.
 *
 * @param x int x coordinate
 * @param y int y coordinate
 *
 * @return Point a Point object
 */
public Point initPoint(int x, int y){
    System.out.println(" in initPoint(int, int)");
    this.x = x;
    this.y = y;
    return this;
}

/**
 * move a point
 *
 * @param x int delta x value
 * @param y int delta y value
 *
 * @return Point a Point object
 */
public Point move(int x, int y){
    System.out.println(" in move(int, int)");
    this.x += x;
    this.y += y;
    return this;
}

/**
 * Returns the x coordinate of a point
 *
 * @return int x value
 */
public int getX(){
    System.out.println(" ingetX()");
    return this.x;
}

/**
 * Returns the y coordinate of a point
 *
 * @return int y value
 */
public int getY(){
    System.out.println(" in getY()");
    return this.y;
}
06     * @return int x value
07     */
08     public int getY()
09     {
10         System.out.println(" in getY()");
11         return this.y;
12     }
13     */
14     * Returns how many points are created so far.
15     *
16     * @return int nPoints
17     */
18     public int getNPoints()
19     {
20         System.out.println(" in getNPoints()");
21         return this.nPoints;
22     }
23

Source Code: Src/5/Point.java

II. Execution of the test program:

Point.soManyPoints = 0
    in Point() constructor
    in Point(int, int) constructor
    in getX()
    x = 2
    in getY()
    y = 3
    in Point() constructor
    in initPoint(int, int)
    in getX()
    x = 4
    in getY()
    y = 5
    in move(int, int)
    in getX()
    x = 10
    in getY()
    y = 12
    in getNPoints()
    nPoints = 2
    aPoint.soManyPoints = 2

You may find the javadoc pages
6.11. Additional Examples

See

```java
public class Scope_1 {
  String aString = null;

  public void method(String aString) {
    this.aString = new String("set in method");
    System.out.println("2. method:aString:" + this.aString);
  }

  public void test() {
    String aString = new String("set in test");
    System.out.println("1. test:aString:" + aString);
    method(aString);
    System.out.println("3. test:aString:" + aString);
  }

  public static void main(String args[]) {
    new Scope_1().test();
  }
}
```

Source Code: Src/5/Scope_1.java

```java
public class Scope_2 {
  String aString = null;

  public void test() {
    String aString = new String("set in test");
    // is there a way that "set in method"
    System.out.println("1. test:aString:" + aString);
    method(aString);
    System.out.println("3. test:aString:" + aString);
  }

  public static void main(String args[]) {
    new Scope_2().test();
  }
}
```

Source Code: Src/5/Scope_2.java

```java
public class Scope_3 {
  String aString = null;

  public void test() {
    int i;
```
class Test {
    int i;
}

for (int index = 0; index < 10; index++) {
    System.out.println("index = " + index);
}

public static void main(String args[]) {
    new Scope_3().test();
}

Source Code: Src/5/Scope_3.java

// see http://docs.oracle.com/javase/specs/jls/se7/html/jls-6.html#jls-6.3 example 6.4-1
public class Scope_4 {
    String aString = null;

    public static void test_2() {
        int k = 0;
        for (int index = 0; index < 10; index++) {
            int k = 3;
        }
    }

    public static void test() {
        int i;
        int k = 0;
        switch (k) {
            case 1:
                int i = 1;
                System.out.println("1: i == " + i);
            }
            break;
            default:
                System.out.println("something went wrong!");
                break;
        }
    }

    public static void main(String args[]) {
        test();
    }
}

Source Code: Src/5/Scope_4.java


Newly declared fields can hide fields declared in a superclass or super interface. Newly declared
methods can hide, implement, or override methods declared in a superclass. Would hiding just mean that new declared things would be accessed first, because we still can access the superclass members by means of (super.field) and/or (super.method(arg))?

- Static declaration inside a method change the lifetime of a variable. Can static variables be declared inside of a method? How does the JVM find the variable aStaticLocalVariable?(Question from the second link)

- System.out.println("om: " + this); What kind of address is this? -> om: X@1ad086a Is it the value of the reference to this object? Is it some virtual address provided by the JVM?

- What are transient and volatile variables? When are they used? If I assign a value to the final variable at the time of declaration but I don’t declare it to be static, will still there be copies of this variable with the objects or this variable will behave like a static final?

- This program is confusing for me. I am not able to comprehend the concept that it introduces.

- Some confusions regarding scopes.

```java
void func(){
    int index=2;
    for (int index=0; index<5; index++) {//some code}
}
```

It doesn’t compile saying that the variable "index" is already defined.

```java
void func1(){
    for(int index=0; index<5; index++){//some code}
    System.out.println(index); // Does not compile saying that the variable "index" cannot
} //some code
```

Similar thing happens in switch case. I also read somewhere that hiding of parameter inside the method body is illegal. What happens in the case of inner classes?

- `s.findInLine("(\d+) fish (\d+) fish (\w+) fish (\w+)"`;
Which of the following method executes it?

```
String findInLine(P attern pattern) OR String findInLine(String pattern)
```

Is it a regular expression? If yes, why is the argument using the delimiter to be " fish "? It should have taken the entire thing as one.

- How does the auto-boxing actually proceeds in the following program from this link:-

```
public class SimpleBoxingI {
    public static void main( String[] args ) {
        Integer n1 = new Integer( 42 );
        int i1 = n1;
        int i2 = 31;
        Integer n2 = i1 + i2 + n1; // are all of them boxed?
        System.out.println(n1 + " " + n2 + " " + i1 + " " + i2 );
    }
}
```

One more question regarding boxing:-
class StringToInt {
    public static void main(String args[]) {
        int i;
        Integer aInt = new Integer("4");
        i = aInt;
        //Will the difference between the line above and the line below be auto unboxing and unboxing done by a method respectively?
        i = aInt.intValue();
        i = Integer.parseInt("4");
    }
}

public class SimpleBoxingTypes {
    public static void main(String[] args) {
        Float f1 = new Float(42F);
        Integer i1 = new Integer(42);
        Double d1 = new Double(42.0);

        double f = f1 + (double)i1 + (double)d1;
        //float ff = f1 + (float)i1 + ((float)d1);

        System.out.println(f);
    }
}

Boxing - primitive to Object type
Unboxing - Object to primitive type
What is the concept I am missing in this example? I see only type casting.
Secondly, why does the commented float line doesn’t compile? Why does it start compiling when I use the
primitive types instead of object types? Why do the boxing and unboxing things don’t help? Does this
demonstrate some type of a difference between the two types?
• I see a break statement at the end of the default case everywhere? Does anything significantly change
if I don’t provide it.

• Each parameter specifier consists of a type and an identifier (optionally followed by brackets) that
specifies the name of the parameter.
Where can we use brackets?

7. Inheritance

See also Java has simple inheritance, i.e., a class can only extend one superclass.
Class Members
The members of a class type are all of the following:
• Members inherited from its direct superclass, except in class Object, which has no direct superclass
• Members inherited from any direct super interfaces
• Members declared in the body of the class.
Members of a class that are declared private are not inherited by subclasses of that class. Only members of a class that are declared protected or public are inherited by subclasses declared in a package other than the one in which the class is declared.

Constructors and static initializers are not members and therefore are not inherited.

7.1. Syntax

class subClassName [ extends superClassName ]
{
  ...
}

Super class is object, if extends superClassName is missing.
See
7.2. Example
Remember the Point example.

```java
/**
 * This class implements a point in a three dimensional area.
 * All methods print the method’s name, when they are called.
 *
 * @version $Id$
 * @author hp bischof
 *
 * Revisions:
 * $Log$
 */

public class ThreeDPoint extends Point {
    private int z; // y coordinate of the point

    /**
     * Default Constructor.
     *
     * @return ThreeDPoint a ThreeDPoint object
     */
    public ThreeDPoint(){
        // super();
        System.out.println(" in ThreeDPoint() constructor");
    }

    /**
     * Constructor.
     * initialize x and y values of a point
     *
     * @param x x coordinate
     * @param y y coordinate
     *
     * @return ThreeDPoint a ThreeDPoint object
     */
    public ThreeDPoint(int _x, int _y, int _z){
        super(_x, _y);
        this.z = _z;
        System.out.println(" in ThreeDPoint(int, int) constructor");
    }

    /**
     * initializes x and y of a point.
     *
     * @param x int x coordinate
     * @param y int y coordinate
     *
     * @return ThreeDPoint a ThreeDPoint object
     */
    public ThreeDPoint(int _x, int _y){
        super(_x, _y);
        System.out.println(" in ThreeDPoint(int, int) constructor");
    }

    /**
     * initializes x and y of a point.
     *
     * @param x int x coordinate
     * @param y int y coordinate
     *
     * @return ThreeDPoint a ThreeDPoint object
     */
    public ThreeDPoint(int _x){
        super(_x);
        System.out.println(" in ThreeDPoint(int) constructor");
    }
```
public ThreeDPoint initThreeDPoint(int _x, int _y, int _z) {
    System.out.println(" in initThreeDPoint(int, int) ");
    this.initPoint(_x, _y);
    this.z = _z;
    return this;
}

/**
 * moves a point
 *
 * @param _x int delta x value
 * @param _y int delta y value
 *
 * @return ThreeDPoint a ThreeDPoint object
 */
public void move(int _x, int _y, int _z) {
    System.out.println(" in ThreeD move(int, int, int) ");
    this.move(_x, _y);
    this.z += _z;
}

/**
 * Returns the z coordinate of a point
 *
 * @return int z value
 */
public int getZ() {
    System.out.println(" in getZ() ");
    return this.z;
}

Source Code: Src/6/ThreeDPoint.java
public class TestThreeDPoint {
    private static ThreeDPoint aPoint;
    private static Point twoDPoint;

    public static void main(String args[]) {
        aPoint = new ThreeDPoint(2, 3, 4);
        System.out.println("x = " + aPoint.getX());
        System.out.println("y = " + aPoint.getY());
        System.out.println("z = " + aPoint.getZ());

        aPoint = new ThreeDPoint();
        aPoint.initThreeDPoint(4, 5, 6);
        System.out.println("x = " + aPoint.getX());
        System.out.println("y = " + aPoint.getY());
        System.out.println("z = " + aPoint.getZ());

        aPoint.move(6, 7, 1);
        System.out.println("x = " + aPoint.getX());
        System.out.println("y = " + aPoint.getY());
        System.out.println("z = " + aPoint.getZ());

        System.out.println("nPoints = " + aPoint.getNPoints());

        twoDPoint = aPoint;
        // aPoint = twoDPoint;

        /*
         * Incompatible type for =.
         * Explicit cast needed to
         * convert Point to ThreeDPoint.
         */
    }
}

Source Code: Src/6/TestThreeDPoint.java

Output:

% java TestThreeDPoint
    in Point() constructor
    in Point(int, int) constructor
    in ThreeDPoint(int, int) constructor
    in getX()
    x = 2
    in getY()
    y = 3
    in getZ()
    z = 4
    in Point() constructor
    in ThreeDPoint() constructor
    in initThreeDPoint(int, int)
    in initPoint(int, int)
    in getX()
x = 4
  in getY()
y = 5
  in getZ()
z = 6
  in ThreeD move(int, int, int)
  in move(int, int)
  in getX()
x = 10
  in getY()
y = 12
  in getZ()
z = 7
  in getNPoints()
nPoints = 2

7.3. Constructor Sequence

```java
public class C1 {
    public C1() {
        System.out.println(" in C1");
    }
    public C1(int x) {
        System.out.println(" in C1\!int x");
    }
}
```

Source Code: Src/6/C1.java

```java
public class C2 extends C1 {
    public C2() {
        System.out.println(" in C2");
    }
    public C2(int x) {
        super(x);
        System.out.println(" in C2\!int x");
    }
    public static void main(String args[]) {
        System.out.println("new C1() ... ");
        new C1();
        System.out.println("new C2() ... ");
        new C2();
        System.out.println("new C2(int x) ... ");
        new C2( 3 );
    }
}
```
Source Code: Src/6/C2.java

```java
% java C2
new C1() ...
in C1
new C2() ...
in C1
in C2
new C2(int x) ...
in C1
in C2!int x
```

7.4. How to get access to super class methods/variables?

- Super class:

```java
public class S {
    public int intS; // what is the value of intS?

    public S () {
        System.out.println("in S constructor");
    }

    public S method(int x) {
        intS = x;
        System.out.println("in S!ups");
        return this;
    }

    public String toString() {
        return "S: " + intS;
    }

    public static void main(String args[]) {
        System.out.println("new S() " + new S());
    }
}
```

Source Code: Src/6/S.java

- Sub class:

```java
public class SubclassOfS extends S {
    public int intS;

    public SubclassOfS () {
        System.out.println("in SubclassOfS constructor");
    }

    public S method(int x) {
        intS = x;
        System.out.println("in SubclassOfS!method");
        super.method(9);
        System.out.println("4. super: " + super.toString());
    }
```
14     super.intS = 4;
15     System.out.println("5. super: " + super.toString());
16     return (S) this;
17 }
18 }
19 public String toString() {
20     return "SSubclassOfS: " + intS;
21 }
22
23 public static void main(String args[]) {
24     S aSubclassOfS = new SubclassOfS();
25     S aS = aSubclassOfS.method(42);
26     // System.out.println(aS);
27     // System.out.println(aSubclassOfS);
28     System.out.println("1. SubclassOfS!intS = "
29     + aSubclassOfS.intS);
30     System.out.println("2. ((S)SubclassOfS)!intS = "
31     + ((S)aSubclassOfS).intS);
32     // method(3); // <--- what is the problem here ...
33 }
34 }
- Private methods and variables are not accessible to subclasses.
- A final class cannot be subclassed.
7.7. Packages

- A package is a set of related classes
- Classes in the same package can access each other’s protected members.
- How to create:
  
  ```java
  package name;
  ```

Show Example in grapecluster ....

7.8. Polymorphism

A subclass can override an inherited method by providing a new, identical method declaration.

```java
/**
 * This class shows how to override a method.
 * @version $Id$
 * @author hp bischof
 * Revisions: $Log$
 */

public class OverWriteTop {
    public static int var;
    
    public void both(int x) {
        var = x;
        System.out.println(" in OverWriteTop!both");
    }
    
    public void notBoth(int x) {
        var = x;
        System.out.println(" in OverWriteTop!notBoth");
    }
    
    Source Code: Src/6/OverWriteTop.java
```
/**
 * This class shows how polymorphism can be used.
 * @version $Id$
 * @author hp bischof
 * Revisions:
 * $Log$
 */

public class OverWrite extends OverWriteTop {
    public static int var;
    
    public void both(int x) {
        var = x;
        System.out.println(" in OverWrite!both");
    }

    public static void main(String args[]) {
        OverWrite aOverWrite = new OverWrite();
        aOverWrite.notBoth(42);
        aOverWrite.both(84);
        System.out.println("OverWrite.var = " + OverWrite.var);
        System.out.println("OverWriteTop.var = " + OverWriteTop.var);
    }
}

Source Code: Src/6/OverWrite.java
Result:

```
% java OverWrite
    in OverWriteTop
    in OverWrite
OverWrite.var    = 666
OverWriteTop.var = 42
```
7.9. Which Variable?

```java
public class S1 {
    public static int staticV = 0;
    public int instanceV = 1;

    public static void printsStatic() {
        System.out.println("S1: staticV: " + staticV);
    }

    public void printInstance() {
        System.out.println("S1: instanceV: " + instanceV);
    }

    public static void main(String args[]) {
        printsStatic();
        new S1().printInstance();
    }
}
```

Source Code: Src/6/S1.java

```java
public class S2 extends S1 {
    // public static int staticV = 2;
    public int instanceV = 5;

    public static void printsStatic() {
        System.out.println("S2: staticV: " + staticV);
    }

    public void printInstance() {
        System.out.println("S2: instanceV: " + instanceV);
    }

    public static void main(String args[]) {
        printsStatic();
        new S2().printInstance();
    }
}
```

Source Code: Src/6/S2.java

7.10. Another Example

Test program:
public class TestTwoDThings {
    public static void main(String args[])
    {
        System.out.println("sum = "+sum);
        System.out.println( aCircle.getType() );
        System.out.println( \"\t\" + aCircle.area() );
        System.out.println( aSquare.getType() );
        System.out.println( \"\t\" + aSquare.area() );
        System.out.println( aCube.getType() );
        System.out.println( \"\t\" + aCube.area() );
    }
}

Source Code: Src/6/TestTwoDThings.java
Super Class:

```java
/**
 * Generic two-d class.
 * @version $Id$
 * @author hp bischof
 * Revisions:
 * $Log$
 */

class TwoDThings {
    private String type; // type of the 2-d object
    public int nObjects;

    /**
     * Default Constructor.
     * @return TwoDThings a TwoDThings object
     */
    public TwoDThings(){
        nObjects++;
    }

    /**
     * Constructor.
     * initialize the type of the object
     * @param type the type of the object
     * @return TwoDThings a TwoDThings object
     */
    public TwoDThings(String _type) {
        //this();
        type = _type;
    }

    public TwoDThings setType(String _type) {
        type = _type;
        return this;
    }

    public String getType() {
        return type;
    }
}
```

Source Code: Src/6/TwoDThings.java
Super Class → Square:

```java
/**
 * This class implements a Square class.
 *
 * @version $Id$
 * @author hp bischof
 *
 * Revisions:
 * $Log$
 */

public class Square extends TwoDThings {

    private int length;

    /**
     * Constructor.
     * initialize length and width values of a Square
     *
     * @param _width int _width of the square
     *
     * @return Square a Square object
     */
    public Square(int _length) {
        super("Square");
        length = _length;
    }

    /**
     * Calculates the area of a square.
     *
     * @return area int area of the square
     */
    public int area() {
        return length * length;
    }
}

Source Code: Src/6/Square.java
```
Super Class → Circle:

```java
/**
 * This class implements a Circle class.
 * @version $Id$
 * @author hp bischof
 * Revisions:
 * $Log$
 */

public class Circle extends TwoDThings {

    private int radius;

    /**
     * Constructor.
     * initialize length and width values of a Circle
     * @param _radius int _radius of the circle
     * @return Circle a Circle object
     */
    public Circle(int _radius) {
        super("Circle");
        radius = _radius;
    }

    /**
     * Calculates the area of an square.
     * @return area int area of the square
     */
    public int area() {
        return (int)(Math.PI * radius * radius);
    }

    Source Code: Src/6/Circle.java
```
Super Class → Square → Cube:

```java
public class Cube extends Square {

    /**
     * Constructor.
     * initialize length and width values of a Cube
     *
     * @param _width int _width of the cube
     *
     * @return Cube a Cube object
     */
    public Cube(int _length) {
        super(_length);
        this.setType("Cube");
    }

    /**
     * Calculates the area of a Cube.
     *
     * @return aere int area of the cube
     */
    public int area() {
        return super.area() * 6;
    }
}

Source Code: Src/6/Cube.java
```
7.11. Inner Classes
Since Java version 1.1 inner classes are possible. Inner classes cannot be declared as native, synchronized, transient or volatile.

Example:

```
1    /**
2     * This class implements a inner class.
3     *
4     * @version $Id$
5     *
6     * @author Axel T. Schreiner
7     * @author hp bischof
8     *
9     * Revisions:
10     * $Log$
11     */
12    class InnerClass {
13        static class A {
14            static void hi () {
15                System.err.println("A.hi");
16            }
17        }
18        class B {
19            void hi () {
20                System.err.println("B.hi");
21            }        
22        }
23        void hi () {
24            class C {
25                void hi () {
26                    System.err.println("C.hi");
27                }
28            }
29            Object o = new C () {
30                void hi () {
31                    System.err.println("D.hi");
32                }        
33            };
34            ((C)o).hi(); new C().hi(); new B().hi();
35        }
36        static public void main (String args []) {
37            new InnerClass().hi();
38            A.hi();
39        }
40    }
```

Source Code: Src/5/InnerClass.java
Hi.A is a nested top-level class and could be an interface. A has the same access to things as other classes in a package; more importantly, A can reach all static things in Hi.

Hi.B is a member class and cannot be an interface or contain static methods such as newInstance(). An instance of B can access all members of that instance of Hi which created it; therefore, an instance of B cannot be created in a class method of Hi. If necessary, the prefix Hi.this is used to access a member of Hi from B.

Hi.C is a local class and cannot be an interface. Methods of Hi.C can access all final local variables and parameters of its surrounding method or block and additionally all members (or static members, depending on context) of Hi.

Hi.o is an object of an anonymous class and has the same access as an object of a local class. The class has no names and consequently no constructor; it can be subclassed from an interface.

All inner classes can have their own instance variables.
We will discuss inner classes more at the end of chapter 6.

7.12. Class Cast - 0

```java
public class S3 {
    public int instanceV = 1;
    public void set(int value) {
        instanceV = value;
    }
    public String toString() {
        return "S3: " + instanceV;
    }
    public static void main(String args[]) {
        System.out.println(new S3());
    }
}
```

Source Code: Src/6/S3.java

```java
public class S4 extends S3 {
    public int instanceV = 4;
    public void onlyInS4() {
        System.out.println("S4: onlyInS4");
    }
}
```
```java
public void set(int value) {
    instanceV = value;
}

public String toString() {
    return "S4: " + instanceV;
}

public static void main(String args[]) {
    S4 aS4 = new S4();
    S3 aS3 = (S3)aS4;

    System.out.println("aS4 = " + aS4);
    System.out.println("aS4.instanceV = " + aS4.instanceV);

    System.out.println("aS3 = " + aS3);
    System.out.println("aS3.instanceV = " + aS3.instanceV);

    System.out.println("S4.set(44); ");
    System.out.println("S3.set(33); ");
    aS4.set(44);
    aS3.set(33);

    System.out.println("aS4 = " + aS4);
    System.out.println("aS4.instanceV = " + aS4.instanceV);

    System.out.println("aS3 = " + aS3);
    System.out.println("aS3.instanceV = " + aS3.instanceV);
}

Source Code: Src/6/S4.java

7.13. Class Cast

```
public class S6 extends S5 {
    public int instanceV = 6;
    public void both() {
        instanceV = 100;
    }
    public String toString() {
        return "S6: " + instanceV;
    }
    public int superA() {
        return super.instanceV;
    }
    public static void main(String args[]) {
        S6 aS6 = new S6();
        S5 aS5 = (S5)aS6;
        System.out.println("1. aS6 =\n" + aS6);
        System.out.println("2. aS5 =\n" + aS5);
        System.out.println("3. aS6.superA(): " + aS6.superA());
        // System.out.println("3. aS6.superA(): " + super.S6.instanceV);
    }
}

Source Code: Src/6/S6.java

Output of:
1. aS6 =S6: 6
2. aS5 =S6: 6
3. aS6 =S6: 100
4. aS5 =S6: 100
5. aS6 =S6: 100
6. aS5 =S6: 100
7. aS6.superA(): 1

7.14. Class Cast - II

public class S7 extends S5 {
    public int instanceV = 6;
    public void onlyInS7() {
        instanceV = 100;
    }
    public void both() {
        instanceV = 100;
    }
}
public String toString() {
    return "S7: " + instanceV;
}

public int superA() {
    return super.instanceV;
}

public static void main(String args[]) {
    S7 aS7 = new S7();
    S5 aS5 = (S5)aS7;
    aS7.both();
    aS5.onlyInS7();
}

Source Code: Src/6/S7.java

S7.java:24: cannot find symbol
symbol : method onlyInS7()
location: class S5
    aS5.onlyInS7();
    ^
1 error

shell returned 1

7.15. Abstract Classes

An abstract class

• specifies a public method interface which can be inherited by direct or indirect subclasses.
• may declare methods, but not implement them.
• cannot be instantiated.

Classes who extend an abstract class share the same, possibly extended, interface.

Use:

public class TestAbstract {

    public static void main(String args[])
    {
        Square aSquare = new Square(2);
        Circle aCircle = new Circle(2);
        System.out.println( "Circle" );
        System.out.println( "\t" + aCircle.area() );
        System.out.println( "\t" + aCircle.perimeter() );
    }

}
System.out.println( "Square" );
System.out.println( "\t" + aSquare.area() );
System.out.println( "\t" + aSquare.perimeter() );

Source Code: Src/6b/TestAbstract.java
abstract class Area extends Object {
    String type;
    public String getType() {
        return type;
    }
    public abstract int area();
    public abstract int perimeter();
}

Source Code: Src/6b/Area.java
This class implements a Circle class.

@version $Id$
@author hp bischof

Revisions:
$Log$

public class Circle extends Area {
    private int radius;
    public Circle(int _radius) {
        type = "Circle";
        radius = _radius;
    }
    public int area() {
        return (int)(Math.PI * radius * radius);
    }
    public int perimeter() {
        return (int)(Math.PI * radius * radius);
    }
}

Source Code: Src/6b/Circle.java
You will get a compiler error, if a class doesn’t implement all methods.

% javac C*a
Circle.java:12: class Circle must be declared abstract.
It does not define int perimeter() from class Area.
public class Circle extends Area {
  
1 error
public class Square extends Area {

    private int length;

    public Square(int _length) {
        type = "Square";
        length = _length;
    }

    public int area() {
        return length * length;
    }

    public int perimeter() {
        return 4 * length;
    }
}

Source Code: Src/6b/Square.java

7.16. Use in an Array

Area allTwoDThings[] = new Area[MAXIMUM];

public class TestAbstract_2 {
    static final int MAXIMUM = 4;

    public static void main(String args[]) {
        Area allTwoDThings[] = new Area[MAXIMUM];
        for ( int i = 0; i < MAXIMUM; i++ ) {
            if ( i % 2 == 0 )
                allTwoDThings[i] = new Square(2 * ( i + 24 ));
            else
                allTwoDThings[i] = new Circle(2 * ( i + 24 ));
        }
        int sumOfAllAreas = 0;
for (int i = 0; i < MAXIMUM; i++ )
    sumOfAllAreas += allTwoDThings[i].area();
System.out.println("sumOfAllAreas = " + sumOfAllAreas );
}

7.17. Class Cast and Abstract Classes

abstract class A {
    public int x;
    abstract public A a(int x);
    public A aa(int x) {
        System.out.print("- in A!aa");
        return this;
    }
}

class AX extends A {
    public int x;
    public A a(int x) {
        System.out.print("= in AX!a");
        return this;
    }
    public static void main(String args[]) {
        AX aAX = new AX();
        A aA = (A)aAX;
        System.out.println("aAX.a(42) " + aAX.a(42) );
        System.out.println("aAX.a(43) " + aAX.aa(43) );
        System.out.println("aA.aa(44) " + aA.aa(44) );
        System.out.println("aA.a(45) " + aA.a(45) ); // <--
% java AX
- AX!axAX.a(42) AX@e76cbf7
- AX!aaaAX.a(43) AX@e76cbf7
- AX!aaaA.a(44) AX@e76cbf7
- AX!aaa.a(45) AX@e76cbf7

1 class AXX extends A {
2     public int x;
3     
4     public A a(int x) {
5         System.out.println(" in AX!a");
6         return this;
7     }
8 
9     public static void main(String args[]) {
10         AX aAX = new AX();
11         AXX aAXX = new AXX();
12         
13         System.out.println("aAX.a(42) " + aAX.a(42) );
14         System.out.println("aAXX.a(43) " + aAXX.a(43) );
15     }
16 }
17 }

Source Code: Src/6/AXX.java

7.18. Site Note: Documentation — javadoc

The various JDK packages are documented on HTML pages in a standardized format that contains many references to other classes or replaced methods etc.
creates the documentation directly from the program sources. Special comments /** ... */ before class, variable and method declarations are transferred to the documentation. The comments may contain significant elements:

@see class#method creates a reference.
@param name text describes a method parameter.
@return text describes the result value of a method.
@exception class text describes an exception.

The documentation is useful even without special comments because it describes the embedding in the class hierarchy and the redefinition of methods. References to existing documentation and graphics would have to be post processed, however.
Example:

```bash
% javadoc ----
Usage: javadoc [options] [packagenames] [sourcefiles] [@files]
- overview <file> Read overview documentation from HTML file
- public Show only public classes and members
- protected Show protected/public classes and members (default)
- package Show package/protected/public classes and members
- private Show all classes and members
- help Display command line options
- doctlet <class> Generate output via alternate doclet
- docletpath <path> Specify where to find doclet class files
- 1.1 Generate output using JDK 1.1 emulating doclet
- sourcepath <pathlist> Specify where to find source files
- classpath <pathlist> Specify where to find user class files
- bootclasspath <pathlist> Override location of class files loaded
    by the bootstrap class loader
- extdirs <dirlist> Override location of installed extensions
- verbose Output messages about what Javadoc is doing
- locale <name> Locale to be used, e.g. en_US or en_US_WIN
- encoding <name> Source file encoding name
- J<flag> Pass <flag> directly to the runtime system

Provided by Standard doclet:
- d <directory> Destination directory for output files
- use Create class and package usage pages
- version Include @version paragraphs
- author Include @author paragraphs
- splitindex Split index into one file per letter
- windowtitle <text> Browser window title for the documentation
- doctitle <html-code> Include title for the package index(first) page
- header <html-code> Include header text for each page
- footer <html-code> Include footer text for each page
- bottom <html-code> Include bottom text for each page
- link <url> Create links to javadoc output at <url>
- linkoffline <url1> <url2> Link to docs at <url1> using package list at <url2>
- group <name> <p1>:<p2>.. Group specified packages together in overview page
- nodeprecated Do not include @deprecated information
- nodeprecatedlist Do not generate deprecated list
- notree Do not generate class hierarchy
- noindex Do not generate index
- nohelp Do not generate help link
- navbar Do not generate navigation bar
- helpfile <file> Include file that help link links to
- stylesheetfile <path> File to change style of the generated documentation
- docencoding <name> Output encoding name

1 error

% javadoc -d Html TestTwoDThings.java Square.java Circle.java \
   Cube.java TwoDThings.java
Loading source file TestTwoDThings.java...
Loading source file Square.java...
Loading source file Circle.java...
```
Loading source file Cube.java...
Loading source file TwoDThings.java...
Constructing Javadoc information...
Building tree for all the packages and classes...
Building index for all the packages and classes...
Generating Html/overview-tree.html...
Generating Html/index-all.html...
Generating Html/deprecated-list.html...
Building index for all classes...
Generating Html/allclasses-frame.html...
Generating Html/index.html...
Generating Html/packages.html...
Generating Html/Circle.html...
Generating Html/Cube.html...
Generating Html/Square.html...
Generating Html/TestTwoDThings.html...
Generating Html/TwoDThings.html...
Generating Html/serialized-form.html...
Generating Html/package-list...
Generating Html/help-doc.html...
Generating Html/stylesheet.css...

You may find the documentation here:
The makefile which I used to create the javadoc for chapter 5/Testpoint:

```
1 2CLASSPATH = .# default explicit classpath
3  C = .class# class files
4  J = .java# Java source files
5  JAR = jar
6  JAVA = CLASSPATH=$(CLASSPATH) java
7  JAVAC = CLASSPATH=$(CLASSPATH) javac
8  JAVADOC = CLASSPATH=$(CLASSPATH) javadoc
9  JDOC = javadoc
10
11all:: $C
12
13 1: $C
14 $(JAVA) Expression
15 2: $C
16 @ $(JAVA) Expression -c
17 3: $C
18 $(JAVA) Go
19
20 jdoc:
21 if [ ! -d $(JDOC) ]; then mkdir $(JDOC); fi
22 $(JAVADOC) \
23 -d $(JDOC) \
24 -use \
25 -splitIndex \
26 -windowtitle 'Expression ' \
27 -doctitle 'LP<sup><font size="-2">TM</font></sup> Expression' \
```
7.19. Additional Examples

Given are the following class hierarchy:

```java
abstract class A {
    public abstract int isAbstract();
    public A concrete() {
        System.out.println("A!concrete()");
        return this;
    }
}
```

Source Code: Src/6_a/A.java

```java
class B extends A {
    public B() {
        System.out.println("B()");
    }
    public int isAbstract() {
        System.out.println("B!isAbstract()");
        return 1;
    }
    public A concrete() {
        System.out.println("B!concrete()");
        return this;
    }
}
```

Source Code: Src/6_a/B.java
class C extends A {

    public C() {
        System.out.println(" C()");
    }

    public int isAbstract() {
        System.out.println(" C!isAbstract()");
        return 2;
    }

    public static void main(String args[]) {
        B aB = new B();
        C aC = new C();

        aB.isAbstract();
        aC.isAbstract();

        (aB.concrete()).isAbstract();
        (aC.concrete()).isAbstract();
    }
}

Source Code: Srpc/6_a/C.java

• Draw the class hierarchy.
% java C
   B()
      C()
         B!isAbstract()
         C!isAbstract()

B!concrete()
          B!isAbstract()

A!concrete()
          C!isAbstract()

• Why what is happening in the marked lines. Will this program compile? Will this program run?

class UseBandC {

    public static void main(String args[])
    {
        int sum = 0;
        final int MAX; // or final int MAX = 6;
        MAX = 3;
        Object[] aArray = new Object[MAX]; // ***
        // A[] aArray = new A[MAX]; // ***
        for (int i = 0; i < aArray.length; i++)
if ( i % 2 == 0 )
    aArray[i] = new B(); // ***
else
    aArray[i] = new C(); // ***

for ( int i = 0; i < aArray.length; i++ )
    sum += aArray[i].isAbstract(); // ***
    // sum += ( (A)aArray[i]).isAbstract(); // ***

% java UseBandC
    B()
    C()
    B()
    B!isAbstract()
    C!isAbstract()
    B!isAbstract()

Why what is happening in the marked lines. Will this program compile? Will this program run?
7.20. Interfaces

- An interface specifies which methods must be implement.
- An interface defines an public API.
- This means, we can make sure, that unrelated classes share the same part of the interface.
- An interface defines constants. They are public, static and final regardless of whether these modifiers have been specified.

```java
public interface InCommon {
    static double MINIMUM_INCREASE = 1.6;
    public void volume();
    public void setPrice(int x);
}
```

Source Code: Src/6c/InCommon.java

```java
public class Phone implements InCommon {
    int price;

    public void volume() {
        System.out.println("Pott! volume: 0.5 l");
    }

    public void setPrice(int x) {
        int letSee = (int)(price * MINIMUM_INCREASE);
        price = letSee > x ? x : letSee;
    }
}
```

Source Code: Src/6c/Phone.java

```java
public class VCR implements InCommon {
    int price;

    public void volume() {
        System.out.println("Mug! volume: 0.3 l");
    }

    public void setPrice(int x) {
        int letSee = (int)(price * MINIMUM_INCREASE);
        price = letSee > x ? x : letSee;
    }
}
```

Source Code: Src/6c/VCR.java
An interface can extend or inherit from more than one interface

```java
public interface C extends A, B {
    int AB = 1;
    // Attempt to reference field AB in a int.
    public void c();
}
```
Source Code: Src/6c/C.java

```java
public interface B {
    static int B = 2;
    int AB = 2;
    public void b();
}
```
Source Code: Src/6c/B.java

```java
public interface A {
    static int A = 1;
    int AB = 1;
    public void a();
}
```
Source Code: Src/6c/A.java

```java
public class Cuse implements C {
    public void a() {
        System.out.println("CUse!a");
        // System.out.println("B = " + A.AB);
    }
    public void b() {
        System.out.println("CUse!b");
    }
    public void c() {
        System.out.println("CUse!c");
    }
    public static void main(String argv[]) {
        new Cuse().a();
        System.out.println("A = " + A);
        System.out.println("B = " + B);
    }
}
```
• Interface methods can’t be native, static, synchronized, final, private, or protected
• Abstract and native methods can’t have a body.
• Fields in a field a static and final.

```java
19 }
20 }
21
Source Code: Src/6c/Cuse.java

• Interface methods can’t be native, static, synchronized, final, private, or protected
• Abstract and native methods can’t have a body.
• Fields in a field a static and final.

1 public interface X {
2
3     static double MINIMUM_INCREASE = 1.6;  // % final
4
5     /*
6     * Interface methods can’t be native,
7     * static, synchronized, final, private, or protected
8     Abstract and native methods can’t have a body.
9     */
10     public void volume()
11     {
12         System.out.println("xxxx");
13     }
14     public void setPrice(int x);
15 }
Source Code: Src/6c/X.java
```
7.21. Aggregation

- Aggregation is a design term, which means that you create a new object by composing it out of the others.
- Aggregation relationships are specified by classes and reflected by their instance objects.
- For example: A Cylinder class can be defined as:

```java
/**
 * This class implements a Cylinder Class
 * NOT COMPLETE
 * @version $Id$
 * @author hp bischof
 * Revisions:
 * $Log$
 */

public class Cylinder {
    private aCircle;
    private aRect;

    public Cylinder(int _radius, _height) {
        aCircle = new Circle(radius);
        aRect = new Rectangle(aCircle.perimeter(), height);
    }

    public int area() {
        return aCircle.area * 2 + aRect.area();
    }

    ....
}

Source Code: Src/6b/Cylinder.java
```
7.22. Short Examples for Clarification

Default Constructor Sequence
Which constructor is called when?

```java
1  public class X_1 {
2     
3     public X_1() {
4         System.out.println(" in X_1!X_1()");
5     }
6     
7     public X_1(int x) {
8         System.out.println(" in X_1!X_1(int x)");
9     }
10    
11    public X_1(int x, int y) {
12        System.out.println(" in X_1!X_1(int x, int y)");
13    }
14 
15 }
```

Source Code: Src/6g/X_1.java
class X_2 extends X_1 {

    public X_2() {
        // super();  // default
        System.out.println(" in X_2!X_2()");
    }

    public X_2(int x) {
        // super();  // default
        super(x);
        System.out.println(" in X_2!X_2(int x)"');
    }

    public X_2(int x, int y) {
        // super();  // default
        System.out.println(" in X_2!X_2(int x, int y)"');
    }

    public static void main(String args[]) {
        X_2 aX_2 = new X_2();
        X_2 aaX_2 = new X_2(3);
        X_2 aaaX_2 = new X_2(3, 3);
    }

    Source Code: Src/6g/X_2.java

Result:

    in X_1!X_1()
    in X_2!X_2()
    in X_1!X_1(int x)
    in X_2!X_2(int x)
    in X_1!X_1()
    in X_2!X_2(int x, int y)
Constructor must match
Superclass has no default() constructor!

```java
1  public class X_1 extends Object {
2
3    public X_1() {
4      System.out.println(" in X_1!X_1()");
5    }
6    public X_1(int x) {
7      System.out.println(" in X_1!X_1(int x)");
8    }
9
10  }
```

Source Code: Src/6h/X_1.java
class X_2 extends X_1 {
  /*
   * public X_2() {
   * System.out.println(" in X_2!X_2() ");
   */
  
  public static void main(String args[]) {
    X_2 aX_2 = new X_2();
  }
}

Source Code: Src/6h/X_2.java

Result:
% javac X*a
X_2.java:3: No constructor matching X_1() found in class X_1.
   public X_2() {
    ^
1 error

- Overloading of constructors is identical in behavior to overloading of methods. The overloading is resolved at compile time by each class instance creation expression.
- If a class contains no constructor declarations, then a default constructor that takes no parameters is automatically provided:
- If the class being declared is the primordial class Object, then the default constructor has an empty body.
- Otherwise, the default constructor takes no parameters and simply invokes the superclass constructor with no arguments.
- A compile-time error occurs if a default constructor is provided by the compiler but the superclass does not have a constructor that takes no arguments.
Methods

Access of methods and super methods.

```java
public class X_1 {
    public X_1() {
        System.out.println(" in X_1!X_1() ");
    }

    public X_1(int x) {
        System.out.println(" in X_1!X_1(int x) ");
    }

    public void a() {
        System.out.println(" in X_1!a() ");
    }
}
```

Source Code: Src/6f/X_1.java
class X_2 extends X_1 {
    public X_2() {
        super();  // default
        super.a();
        System.out.println(" in X_2!X_2()");
    }

    public X_2(int x) {
        super();  // default
        super(x);
        System.out.println(" in X_2!X_2(int x)");
    }

    public void a() {
        super.a();
        System.out.println(" in X_2!a()");
    }

    public static void main(String args[]) {
        X_2 aX_2 = new X_2();
        aX_2.a();
        X_2 anOtherX_2 = new X_2(3);
    }
}

Source Code: Src/6f/X_2.java

Result:
% java X_2
    in X_1!X_1()
    in X_1!a()
        in X_2!X_2()
    in X_1!a()
    in X_2!a()
Instance Variables
Which one do I get?

```java
public class X_1 extends Object {
    int x1 = 1;
    int x2 = 11;
}
```

Source Code: Src/6i/X_1.java
```java
class X_2 extends X_1 {
    int x1 = 2;

    public void a() {
        System.out.println(" in X_1!a()");
        System.out.println(" in X_1!a()!x1 = " + x1 );
        System.out.println(" in X_1!a()!(X_1)this.x1 = " + this.x1 );
        System.out.println(" in X_1!a()!super.x1 = " + super.x1 );
    }

    public static void main(String args[])
    {
        X_2 aX_2 = new X_2();
        System.out.println(" main!x1 = " + aX_2.x1 );
        aX_2.a();
    }
}
```

Source Code: Src/6i/X_2.java

Result:
```
% java X_2
main!x1 = 2
   in X_1!a()
   in X_1!a()!x1 =
   in X_1!a()!(X_1)this.x1 =
   in X_1!a()!super.x1 =
```
Private or Protected?
Can I?

```
1   public class X_1 extends Object {
2       static int x1 = 1;
3       private int x2 = 11;
4   }
5
Source Code: Src/6k/X_1.java
```
class X_2 extends X_1 {
    static int x1 = 2;
    int x2 = 22;

    public void a() {
        System.out.println(" in X_1!a()");
        System.out.println(" in X_1!a()!x1 = " + x1);
        System.out.println(" in X_1!a()!super.x1 = "
                           + super.x1);
        System.out.println(" in X_1!a()!X_1.x1 = "
                           + X_1.x1);
        /* System.out.println(" in X_1!a()!super.x2 = "
                           + super.x2);
        * X_2.java:13: Variable x2 in class
        * X_1 not accessible from class X_2.
        */
    }

    public static void main(String args[]) {
        X_2 aX_2 = new X_2();
        System.out.println(" main!x1 = " + aX_2.x1);
        System.out.println(" main!x1 = " + aX_2.x2);
        aX_2.a();
    }
}

Source Code: Src/6k/X_2.java
Result:

```
% java X_2
  main!x1 = 2
  main!x1 = 22
  in X_1!a()
  in X_1!a()!x1 = 2
  in X_1!a()!super.x1 = 1
  in X_1!a()!X_1.x1 = 1
```
7.23. A Binary Search Tree

Interface:

```java
public interface Node {

    /**
     * Returns true if this Node is less than the Node referred to by the
     * argument.
     *
     * @param Node the node to compare this node with
     *
     * @return true if the this Node is less than the Node argument.
     */
    abstract public boolean isLess(Node aNode);

    /**
     * Returns true if this Node is equal to the Node referred to by the
     * argument.
     *
     * @param Node the node to compare this node with
     *
     * @return true if the this Node is equal to the Node argument.
     */
    abstract public boolean isEqual(Node aNode);

    /**
     * Returns true if this Node is greater than the Node referred to by the
     * argument.
     *
     * @param Node the node to compare this node with
     *
     * @return true if the this Node is greater than the Node argument.
     */
    abstract public boolean isGreater(Node aNode);

} // Node
```
Source Code: Src/6t/Node.java
Implementation I:

```java
/**
 * This class allows nodes in a binary tree to hold strings. Since
 * it is an extension of the Node class, these objects can compare
 * themselves to similar objects.
 *
 * Note that the methods in this class take Node objects as a
 * parameter. The method then casts the Node reference to a
 * reference to a StringNode. Since the caller may pass a reference
 * to a Node object that cannot be converted to a StringNode, each of
 * methods in this class may throw a CastClassException.
 *
 * @version $Id$
 *
 * @author Hans-Peter Bischof
 * @author Paul Tymann
 *
 * Revisions:
 * $Log$
 */

public class StringNode implements Node {

    // The string that contains the data
    private String info;

    /**
     * Create a new StringNode.
     *
     * @param info the string that is to be stored in this node
     */

    public StringNode( String info ) {
        this.info = info;
    }

    /**
     * Returns true if this StringNode is less than the StringNode
     * referred to by the argument.
     *
     * @param StringNode the StringNode to compare this
     * StringNode with.
     *
     * @return true if the this StringNode is less than the
     * StringNode argument.
     *
     * @exception CastClassException if the argument cannot be
     * converted to a StringNode
     */

    public boolean isLess( Node aNode ) {
        return info.compareTo( aNode.info ) < 0;
    }
}
```
return (info.compareTo(aNode.info) < 0);// wrong

/**
 * Returns true if this StringNode is equal to the StringNode
 * referred to by the argument.
 *
 * @param StringNode the StringNode to compare this
 * StringNode with.
 *
 * @return true if the this StringNode is equal to the
 * StringNode argument.
 *
 * @exception CastClassException if the argument cannot be
 * converted to a StringNode
 */

public boolean isGreater(Node aNode) {
    StringNode aStringNode = (StringNode)aNode;

    return (info.compareTo(aStringNode.info) > 0);
}

/**
 * Returns true if this StringNode is greater than the StringNode
 * referred to by the argument.
 *
 * @param StringNode the StringNode to compare this
 * StringNode with.
 *
 * @return true if the this StringNode is greater than the
 * StringNode argument.
 *
 * @exception CastClassException if the argument cannot be
 * converted to a StringNode
 */

public boolean isEqual(Node aNode) {
    StringNode aStringNode = (StringNode)aNode;

    return (info.compareTo(aStringNode.info) == 0);
}

/**
 * Return a string representation of the data contained in this
 * StringNode.
 *
 * @return a string representation of this StringNode.
 */

public String toString() {
    return (info);
}
06
07 } // StringNode

Source Code: Src/6t/StringNode.java
Implementation II:

```java
/**
 * This class allows nodes in a binary tree to hold strings. Since
 * it is an extension of the Node class, these objects can compare
 * themselves to similar objects.
 *
 * Note that the methods in this class take Node objects as a
 * parameter. The method then casts the Node reference to a
 * reference to a IntegerNode. Since the caller may pass a reference
 * to a Node object that cannot be converted to a IntegerNode, each of
 * methods in this class may throw a CastClassException.
 *
 * @version $Id$
 * @author Hans-Peter Bischof
 * @author Paul Tymann
 *
 * Revisions:
 * $Log$
 */

class IntegerNode implements Node {
    // The string that contains the data
    private Integer info;

    /**
     * Create a new IntegerNode.
     *
     * @param info the string that is to be stored in this node
     */
    public IntegerNode( int info ) {
        this.info = new Integer (info);
    }

    /**
     * Returns true if this IntegerNode is less than the IntegerNode
     * referred to by the argument.
     *
     * @param IntegerNode the IntegerNode to compare this
     * IntegerNode with.
     *
     * @return true if the this IntegerNode is less than the
     * IntegerNode argument.
     *
     * @exception CastClassException if the argument cannot be
     * converted to a IntegerNode
     */
    public boolean isLess( Node aNode ) {
```
IntegerNode aIntegerNode = ( IntegerNode )aNode; // correct

return ( info.compareTo( aIntegerNode.info ) < 0 );
}

/**
 * Returns true if this IntegerNode is equal to the IntegerNode
 * referred to by the argument.
 *
 * @param IntegerNode the IntegerNode to compare this
 * IntegerNode with.
 *
 * @return true if the this IntegerNode is equal to the
 * IntegerNode argument.
 *
 * @exception CastClassException if the argument cannot be
 * converted to a IntegerNode
 */

public boolean isEqual( Node aNode ) {
    IntegerNode aIntegerNode = ( IntegerNode )aNode;

    return ( info.compareTo( aIntegerNode.info ) == 0 );
}

/**
 * Return a string representation of the data contained in this
 * IntegerNode.
 *
 * @return a string representation of this IntegerNode.
 */

public String toString() {
    return ( info.toString() );
Get the name of a Class:

```java
import java.lang.reflect.Method;

public class PrintClassName {
    public static void printClassName(Object o) {
        System.out.println(o.getClass().getName() +
                           o.getClass().getName());
    }

    public static void printMethods(Object o) {
        Method[] m = o.getClass().getMethods();
        for (int i = 0; i < m.length; i++)
            System.out.println(m[i].getName());
    }

    public static void main(String args[]) {
        String aString = "aaa";
        Integer aInteger = new Integer("0");
        printClassName((Object)aString);
        printClassName((Object)aInteger);
        printMethods((Object)aInteger);
    }
}
```

### 7.24. Nested Classes

- Nested classes can be: non-static or static nested.
- Non-static nested classes are referred to as inner classes.
- A nested class is a class that is a member of another class.

```java
class Outer{
    
    class AnestedClass {
        
    }
}
```

- A nested class can be declared static or not.
- A static nested class is called a static nested class.
A non static nested class is called an inner class.

As with static methods and variables, a static nested class is associated with its enclosing class.

And like class methods, a static nested class cannot refer directly to instance variables or methods defined in its enclosing class—it can use them only through an object reference.

As with instance methods and variables, an inner class is associated with an instance of its enclosing class and has direct access to that object's instance variables and methods.

Because an inner class is associated with an instance, it cannot define any static members itself.

Example:

```java
public class NestedClassEx {
    public int inNestedClass;

    void inNestedClass() {
        System.out.println("NestedClass!inNestedClass");
        (new AinnerClass()).aInnerClassM2();
    }

    static class AstaticClass {
        static void aStaticClassM1() {
            System.out.println("AstaticClass!aStaticClassM1");
        }
        void aStaticClassM2() {
            System.out.println("AstaticClass!aStaticClassM2");
        }
    }

    class AinnerClass {
        /*
         * static void aInnerClassM1() {
         *     System.out.println("AInnerClass!aInnerClassM1");
         * }
         * NestedClassEx.java:15: inner classes cannot have static declarations
         * static void aInnerClassM1() {
         */

        void aInnerClassM2() {
            System.out.println("AInnerClass!aInnerClassM2");
        }
    }

    public static void main(String args[]) {
        AstaticClass.aStaticClassM1();
        (new AstaticClass()).aStaticClassM2();
        (new NestedClassEx()).inNestedClass();
        // (new AinnerClass()).aInnerClassM2();
    }
}
```

Source Code: Src/6/NestedClassEx.java
7.25. Anonymous Classes

- Inner class can be declared without naming it.
- Anonymous classes can make code difficult to read. Their use should be limited to those classes that are very small (no more than a method or two) and whose use is well-understood (like the AWT event-handling adapter classes).
- Example:

```java
public static void main(String[] args) {
    JFrame f = new JFrame("RectangleDemo");
    f.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
    RectangleDemo controller = new RectangleDemo();
    controller.buildUI(f.getContentPane());
    f.pack();
    f.setVisible(true);
}
```

7.26. Static Initializer Blocks

- Static initializer blocks are primarily used for initialization.
- The code in a static initializer block is executed when the class is initialized.
- A class can have more than one static initializer block.
- S.java:

```java
1 public class S {
2     static public int intS;
3     
4     public S () {
5         System.out.println("in S constructor");
6     }
7     
8     static {
9         System.out.println("S:Static 1");
10     }
11     
12     static {
13         System.out.println("S: Static 2");
14     }
15     
16     public static void main(String args[]) {
17         System.out.println("new S() " + new S());
18     }
19 }
```

Source Code: Src/6_AddOn/S.java
• SubclassOfS.java:

```java
public class SubclassOfS extends S {

    public int intS;

    static {
        System.out.println("SubclassOfS: Static 1");
    }

    public SubclassOfS () {
        System.out.println("in SubclassOfS constructor");
    }

    public static void main(String args[]) {
        System.out.println("In SubClass of S");
        SubclassOfS aSubclassOfS = new SubclassOfS();
    }

    Source Code: Src/6_AddOn/SubclassOfS.java
```
7.27. Questions

- Will it compile:

```
public class Yellow {
    private String yellowPrivate = "yellowPrivate";

    public static void main(String args[]) {
        System.out.println(yellowPrivate);
    }
}
```

Source Code: Src/6_q/Yellow.java

- Will it compile and if yes, what is the output: (eine neue variable wird erzeugt).

```
public class Coke {
    private String cokePrivate = "cokePrivate";
    private String s;
    private static String cokePrivateS = "cokePrivateS";

    public void m() {
        cokePrivate = "java";
    }

    public void change(String cokePrivate) {
        cokePrivate = "hello";
    }

    public void print() {
        System.out.println("1. cokePrivate = " + cokePrivate);
        System.out.println("2. cokePrivateS = " + cokePrivateS);
        System.out.println("----------------------------------");
    }

    public static void main(String args[]) {
        Coke aCoke = new Coke();
        aCoke.m();
        aCoke.print();
        aCoke.change("t");
        aCoke.print();
    }
}
```

Source Code: Src/6_q/Coke.java

- Will it compile: (die antwort zu der frage kann nur verneint werden)

```
public class Red {
    private String redPrivate = "redPrivate";
}
```

Source Code: Src/6_q/Red.java
public class Blue {
    private String bluePrivate = "bluePrivate";

    public boolean isLess(Red aRed) {
        return bluePrivate == aRed.redPrivate;
    }

    public static void main(String[] args) {
        Red aRed = new Red();
        Blue aBlue = new Blue();
        aBlue.isLess(aRed);
    }
}

Source Code: Src/6_q/Blue.java

Will it compile and if yes, what is the output:

public class H {
    private String hPrivate = "hPrivate";
    private static String hPrivateS = "hPrivateS";

    public H(String hPrivate) {
        this.hPrivate = hPrivate;
    }

    public void knete() {
        this = this("RIT");
    }

    public void print(String tag) {
        System.out.println(tag + " hPrivate = " + hPrivate);
    }

    public static void main(String[] args) {
        H aH = new H();
        aH.print("1.");
        aH.knete();
        aH.print("2.");
    }
}

Source Code: Src/6_q/H.java

H.java:10: Invalid left hand side of assignment.
    this = this("RIT");
    ^

H.java:10: Only constructors can invoke constructors.
    this = this("RIT");
    ^

H.java:18: No constructor matching H() found in class H.
    H aH = new H();
    ^

Will it compile and if yes, what is the output:

```java
public class Bauer {
    private String bauerPrivate = "bauerPrivate";
    private static String bauerPrivateS = "bauerPrivateS";

    public Bauer() {
    }

    public Bauer(String bauerPrivate) {
        this.bauerPrivate = bauerPrivate;
    }

    public void knete() {
        this = new Bauer("RIT");
    }

    public static void main(String args[]) {
        Bauer aBauer = new Bauer();
    }
}
```

Source Code: Src/6_q/Bauer.java

Will it compile and if yes, what is the output: (tja, das sollte nicht so schwierig sein)

```java
class Oh {
    public static void intMethod(int intArg) {
        intArg = 22;
    }

    public static void intArrayMethod(int[] intArg) {
        intArg[0] = 22;
    }

    public static void main(String[] args) {
        int intVariable = 2;
        int intArray[] = {2, 2, 2};
        System.out.println("1. " + intVariable);
        intMethod(intVariable);
        System.out.println("2. " + intVariable);
        System.out.println("3. " + intArray[0]);
        intArrayMethod(intArray);
        System.out.println("4. " + intArray[0]);
    }
}
```

Source Code: Src/6_q/Oh.java

Will it compile:
public interface I {
    static int iStatic = 1;
    public int iPublic = 6;
    private int iPrivate = 6;
    protected int iProtected = 6;
}

Source Code: Src/6_q/I.java

- Will it compile:

public interface Ic {
    static int iStatic = 1;
    public int iPublic = 6;
    public void furyo();
}

Source Code: Src/6_q/Ic.java

- Will it compile: (eins ist genug)

public class Ic1 implements Ic {
    public void furyo () {
        iPublic = 4;
        iStatic = 2;
    }
}

Source Code: Src/6_q/Ic1.java

public class Bier {
    private int bier;

    public Bier() {
        bier ++;
    }

    public void print() {
        System.out.println("bier = " + bier);
    }

    public static void main(String args[]) {
        Bier aBier = new Bier();
        for ( int i = 0; i < 1000; i ++ )
            aBier = new Bier();
        aBier.print();
    }
Will it compile and what is the output: (ein Object koennte nicht zugwiesen sein)

```java
public class Wein {
    private int wein;

    public Wein() {
        wein ++;
    }

    public void print() {
        System.out.println("wein = " + wein );
    }

    public static void main(String args[]) {
        Wein aWein;
        for ( int i = 0; i < 1000; i ++ )
            aWein = new Wein();
        aWein.print();
    }
}
```

Will it compile and what is the output:

```java
public class ApfelSaft {
    private int gut;

    public ApfelSaft() {
        gut ++;
    }

    public void print() {
        System.out.println("gut = " + gut );
    }

    public static void main(String args[]) {
        ApfelSaft aApfelSaft = null;
        for ( int i = 0; i < 1000; i ++ )
            aApfelSaft = new ApfelSaft();
        aApfelSaft.print();
    }
}
```

Will it compile and what is the output: (das dritte ist falsch)
public class SEqual {
    public static void main(String args[]) {
        String s = "a";
        String b = "a";

        if ( s == b )
            System.out.println("1. s == b ");
        if ( s.equals(b) )
            System.out.println("1. s.equals(b) ");

        if ( "Furyo" == "Furyo" )
            System.out.println("2. \"Furyo\" == \"Furyo\" ");

        s = new String("a");
        b = new String("a");

        if ( s == b )
            System.out.println("3. s == b ");
        if ( s.equals(b) )
            System.out.println("4. s.equals(b) ");
    }
}

Source Code: Src/6_q/SEqual.java

1. s == b
1. s.equals(b)
2. "Furyo" == "Furyo"
4. s.equals(b)
7.28. What is the following Example doing?

- Example 1:

```java
public abstract class Node extends Number {
    public byte byteValue () {
        return (byte)longValue();
    }

    public short shortValue () {
        return (short)longValue();
    }

    public int intValue () {
        return (int)longValue();
    }

    public float floatValue () {
        return (float)doubleValue();
    }

    protected abstract static class Binary extends Node {
        protected Number left;
        protected Number right;
        protected Binary () {}
        protected Binary (Number left, Number right) {
            this.left = left; this.right = right;
        }
    }

    protected abstract static class Unary extends Node {
        protected Number tree;
        protected Unary (){}
        protected Unary (Number tree) {
            this.tree = tree;
        }
    }

    public static class Add extends Binary {
        public Add (Number left, Number right) {
            super(left, right);
        }
    }
```
public long longValue () {
    return left.longValue() + right.longValue();
}

public double doubleValue () {
    return left.doubleValue() + right.doubleValue();
}

public static class Mul extends Binary {
    public Mul (Number left, Number right) {
        super(left, right);
    }

    public long longValue () {
        return left.longValue() * right.longValue();
    }

    public double doubleValue () {
        return left.doubleValue() * right.doubleValue();
    }
}

public static class Minus extends Unary {
    public Minus (Number tree) {
        super(tree);
    }

    public long longValue () {
        return -tree.longValue();
    }

    public double doubleValue () {
        return -tree.doubleValue();
    }
}

public static void main(String args[]) {
    Node aNode = new Node.Add(new Double(1), new Double(2));
    System.out.println("i) aNode = " + aNode.floatValue() );
    aNode = new Node.Add(
            new Node.Mul( new Double(2), new Double(3)),
            new Node.Mul( new Double(4), new Double(5))
    );
    System.out.println("ii) aNode = " + aNode.floatValue() );
    aNode = new Node.Add( 
            new Double(1),
            new Node.Minus(new Double(2))
    );
    System.out.println("iii) aNode = " + aNode.floatValue() );
}

Source Code: Src/6_AddOn/Node.java
7.29. Additional Questions

7.30. Questions:
Basic Type vs. Object

```java
/*
 * Test.java
 * Doubts regarding array of object creation
 */
class A {
    int x = 1;
}
public class Test {
    public static void main(String[] args) {
        int array_int[];
        array_int = new int[5];
        A cl_a = new A(); // reference to a object
        System.out.println("1 " + cl_a);
        A[] cl_array; // reference to the array
        cl_array = new A[5];
        // cl_array[1]= new A();
        System.out.println("2 " + cl_array);
        System.out.println("3 " + cl_array[1]);
        try {
            System.out.println("4 " + cl_array[1].x);
        } catch (Exception e) {
            System.out.println("5 " + e);
        }
    }
}
```

Source Code: Src/8/Test.java

Given is the following hierarchy:

```java
public interface I1 {}
```
Source Code: Src/6_a/I1.java

```java
public interface I2 {}
```
Source Code: Src/6_a/I2.java

```java
public interface I3 extends I1, I2 {}
```
Source Code: Src/6_a/I3.java
public interface I4 extends I2 {}
Source Code: Src/6_a/I4.java

class C1 implements I3 {}
Source Code: Src/6_a/C1.java

class C2 implements I4 {}
Source Code: Src/6_a/C2.java
class C3 extends C1 implements I4 {
    public static void main(String args[]) {
        C1 aC1 = new C1();
        C2 aC2 = new C2();
        C3 aC3 = new C3();
        I1 aI1 = null; I2 aI2 = null; I3 aI3 = null;
        aC1 = (C1)aC3; // 1
        aI1 = (I1)aC3; // 2
        aI1 = (I1)aC2; // 3
        aI2 = (I2)aC3; // 4
        aI1 = (aI1)aI2; // 5
        aI2 = (I2)aI3; // 6
    }
}

Source Code: Src/6_a/C3.java

Draw a graphical representation of the hierarchy.

Please explain which assignment(s) are legal, and which assignment(s) are not valid.

1. 
2. 
3. 
4. 
5. 

Which assignment(s) will cause a compile time error?
Which assignment(s) will cause a run time error?

8. Generics

See also: and

and stolen from their:

• Generics enable types (classes and interfaces) to be parameters when defining classes, interfaces and methods. Much like the more familiar formal parameters used in method declarations, type parameters provide a way for you to re-use the same code with different inputs. The difference is that the inputs to formal parameters are values, while the inputs to type parameters are types.

8.1. Generic Class Definition

class name<T1, T2, ..., Tn> { .... }

Type parameter names are, by convention, single, uppercase letters. This makes it easy to differentiate between a type variable and a class/interface.

Type Parameter Naming Conventions

E - Element
K - Key
N - Number
T - Type
V - Value
8.2. Old versus New

- **Old:**
  
  ```java
  List myIntList = new LinkedList();
  myIntList.add(new Integer(0));
  Integer x = (Integer) myIntList.iterator().next();
  ```

- **New:**
  
  ```java
  List<Integer> myIntList = new LinkedList<Integer>();
  myIntList.add(new Integer(0));
  Integer x = myIntList.iterator().next();
  ```

8.3. Defining Simple Generics I

```
1 2 3 4 5 6 7 8 9
public interface List<E> {
    void add(E x);
    Iterator<E> iterator();
}
public interface Iterator<E> {
    E next();
    boolean hasNext();
}
```

Source Code: Src/11_jdk15/List.java

8.4. Defining Simple Generics II

- **Node:**
  
  ```java
  1 2 3 4 5 6 7 8
  public class Node<F> {
    public Node( F value, Node<F> next ) {
      this.value = value;
      this.next = next;
    }
    public F value;
    public Node<F> next;
  }
  ```

Source Code: Src/11_jdk15/Node.java

- **The Generic Stack class:**
  
  ```java
  1 2 3 4 5 6 7 8
  class RStack<E> {
    public void push( E x ) {
      head = new Node<E>(
        x, head );
    }
    public E pop() {
      E result = head.value;
      head = head.next;
  }
  ```
9          return result;
10       }
11   private Node<E> head = null;
12   
13   public static void main(String args[]) { 
14       RStack<String> s;
15       s = new RStack<String>();
16       s.push("hello");
17       String w = s.pop();
18   }
19  
Source Code: Src/11_jdk15/RStack.java

•   How about legacy code:

1  class Stack {
2   
3   public static void main(String args[]) { 
4       RStack s;
5       s = new RStack();
6       s.push("hello");
7       String w = (String)s.pop();
8   }
9  
Source Code: Src/11_jdk15/Stack.java

% javac Stack.java
Note: Stack.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details

8.5. Remarks
•   A compile-time only language extension.
•   Parameterized types are NOT macro expanded. i.e., no real instantiation
•   Compiler forgets type parameters after checking. Some usage limitations (arrays, instanceof) due to
lack of guaranteed run-time type knowledge
•   Type safety was primary concern. Readability is usually improved [Doug Lea]
•   JDK 7 or later: type arguments can be empty, if the compiler can determine the type from the context
Box<Integer> integerBox = new Box<>();

8.6. An other Example

static void cleanUp( Collection <String> c ) {
    for ( 
        Iterator <String> i = c.iterator();
        i.hasNext();
    )
        if ((<String> i.next()).length() == 4)
            i.remove();
}
8.7. Multiple Parameter Types

```java
import java.util.*;

public class MultipleTypes<E, V> {
    List<E> data = new ArrayList<E>();
    Vector<V> volume = new Vector<V>();

    public V getV() {
        return volume.elementAt(0);
    }
}
```

Source Code: Src/11_W/MultipleTypes.java

8.8. Remarks II

- No matter to what the type parameters are bound, the code for the generic class is the same.
  - Objects* are assumed at compile time.
  - No basic types allowed (see autoboxing)
- The concept of instantiating a generic class is misleading in Java’s case Is this Legal?

```java
List<String> ls = new ArrayList<String>(); // 1
List<Object> lo = ls; // 2
```

- Is an ArrayList of String objects
- It is not legal, because:
  - `lo.add(new Object());`
  - `String s = ls.get(0); // attempts to assign an Object to a String!`
- Line 2 will cause a compile time error

8.9. Bounded Type Parameters

Bounded Type Parameters allow to restrict the types that can be used as type arguments

```java
public class EvenNumbers<T extends Integer> {
    T n;

    public EvenNumbers(T n) {
        this.n = ( n.intValue() % 2 == 0 ) ? n : null;
    }
}
```

Source Code: Src/11_W/EvenNumbers.java

8.10. Multiple Bounds
8.11. A Tricky Thing

- Why is This Example Illegal?

```java
public class MyVector<E> {
    E[] data = null;
    public MyVector( int size ) {
        data = new E[size];
    }
    public E get( int i ) {
        return data[i];
    }
    public void set( int i, E val ) {
        data[i] = val;
    }
}
```

Source Code: Src/11_W/MyVector.java

- `% javac MyVector.java && java 'echo MyVector.java > .x; sed -e 's/.java//d' .x' MyVector.java:4: generic array creation
data = new E[size];`  
  1 error

```java
import java.lang.reflect.Array;

public class MyVector1<E> {
    E[] data = null;
    public MyVector1( int size ) {
        // data = (E[])new Object[size];
        data = (E[])Array.newInstance(E.class, size);
    }
    public E get( int i ) {
        return data[i];
    }
    public void set( int i, E val ) {
        data[i] = val;
    }
    public <E> E[] getArray(Class<E> aClass, int size) {
        @SuppressWarnings("unchecked")
        E[] arr = (E[])Array.newInstance(aClass, size);
        return arr;
    }
    public <E> E[] getArray(int size) {
        @SuppressWarnings("unchecked")
```
E[] arr = (E[]) Array.newInstance(new Object().getClass(), size);
return arr;
}

public static void main(String args[]) {
    MyVector1 aMyVector1 = new MyVector1(11);
aMyVector1.set(0, "a");
    System.out.println("aMyVector1.get(0): " + aMyVector1.get(0));
    /*
     * Exception in thread "main" java.lang.ClassCastException:
     * [Ljava.lang.Object; cannot be cast to [Ljava.lang.String;
     *   t MyVector1.main(MyVector1.java:31)
     */
    // String[] theStrings = (String[])aMyVector1.getArray( 1 );
}

Source Code: Src/11_W/MyVector1.java

% javac -Xlint MyVector1.java
yVector1.java:6: warning: [unchecked] unchecked cast
found : java.lang.Object[]
required: E[]
    data = (E[])new Object[size];
^ MyVector1.java:7: warning: [unchecked] unchecked cast
found : java.lang.Object[]
required: E[]
    data = (E[])getArray( new Object().getClass(), size );
^ MyVector1.java:24: warning: [unchecked] unchecked call to set(int,E) as a member of the raw type MyVector1
    aMyVector1.set(0, "a");
3 warnings
% java MyVector1
aMyVector1.get(0): a

8.12. A Tricky Thing II
• Corrected Version (gives unchecked cast warning)
```java
public class MyVector2<E> {
    Object[] data = null;
    public MyVector2( int size ) {
        data = new Object[size];
    }
    public E get( int i ) {
        return (E)data[i];
    }
    public void set( int i, E val ) {
        data[i] = val;
    }
    public static void main(String args[]) {
        MyVector2 aMyVector2 = new MyVector2(12);
        aMyVector2.set(0, "a");
        System.out.println("aMyVector2.get(0): " + aMyVector2.get(0));
    }
}
```

Source Code: Src/11_W/MyVector2.java

Note: MyVector2.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

8.13. Restrictions on Generics

Stolen from:

- Cannot Instantiate Generic Types with Primitive Types
- Cannot Create Instances of Type Parameters
- Cannot Declare Static Fields Whose Types are Type Parameters
- Cannot Use Casts or instanceof With Parameterized Types
- Cannot Create Arrays of Parameterized Types
- Cannot Create, Catch, or Throw Objects of Parameterized Types
- Cannot Overload a Method Where the Formal Parameter Types of Each Overload Erase to the Same Raw Type
9. Wildcards

Stolen from above:

In generic code, the question mark (?), called the wildcard, represents an unknown type. The wildcard can be used in a variety of situations: as the type of a parameter, field, or local variable; sometimes as a return type (though it is better programming practice to be more specific). The wildcard is never used as a type argument for a generic method invocation, a generic class instance creation, or a supertype.

Upper BoundWildcard: "? extends E": Denotes a family of subtypes of type Type. This is the most useful wildcard (bounded by its super class E) A method that works on lists of Vector and the subtype of Vector, such as Stack, WList<extends Vector>

- Lower BoundWildcard: "? super E": Denotes a family of super types of type Type (bounded by the subclass E)
  A method that works on lists of Integer and the super types of Integer, such as Integer, Number, and Object: WList<? super Integer>

- Unbound: "?": Denotes the set of all types or any

9.1. Wildcard Example

```java
import java.util.*;

public class WildCard {

    public static void printCollection_2(Collection<?> c) {
        for (Object e : c)
            System.out.println("2: " + e);
    }

    public static void printCollection(Collection c) {
        Iterator i = c.iterator();
        while (i.hasNext()) {
            System.out.println("1: " + i.next());
        }
    }

    public static void main(String args[]) {
        String anArray[] = {"echoes", "Shine", "Tiger" };  
        List l = Arrays.asList(anArray);
        printCollection(l);
        printCollection_2(l);
    }
}
```

Source Code: Src/11_W/WildCard.java

9.2. Bound Wildcards

```
public abstract class Shape {
    public abstract void draw(Canvas c);
}

public class Circle extends Shape {
    private int x, y, radius;
```
public void draw(Canvas c) { ... }

public class Rectangle extends Shape {
    private int x, y, width, height;
    public void draw(Canvas c) { ... }
}

• These classes can be drawn on a canvas:

public class Canvas {
    public void draw(Shape s) {
        s.draw(this);
    }
}

• Any drawing will typically contain a number of shapes.
• Assuming that they are represented as a list, it would be convenient to have a method in Canvas that draws them all:

public void drawAll(List<Shape> shapes) {
    for (Shape s: shapes) {
        s.draw(this);
    }
}

• What we really want is:

public void drawAll(List<? extends Shape> shapes) { ... }

class C<T extends Figure> {...}

— T must descend from Figure.
— Allows Figure’s features to be referenced.
— Keyword extends is even used for interfaces!

• Example:

public void drawAll(List<? extends Shape> shapes) { ... }

• List<? extends Shape> is an example of a bounded wildcard
• However, in this case, we know that this unknown type is in fact a subtype of Shape. Shape is the upper bound of the wildcard.
• This defines upper bounds

9.3. Upper Bound Wildcards

• <? extends E> is called upper bounded wildcard because it defines a type that is bounded by the superclass E
• Collection<? extends Vector> c: Collection of Vector, Stack

1 import java.util.*;
2
3 public class UpperBound {
4
5     public static void printCollection_2(Collection<? extends Vector> c) {
6         }
7     }
8
The type `Collection<? super String>` is a supertype of any `Collection` where `T` is a supertype of `String`.
- This collection can store Strings and Objects.

### 9.4. Lower Bound Wildcards

- `<? super E>` defines a type that is bounded by the subclass `E`.
- `Collection<? super Integer> c`: Collection of Integer, Number, Object

```java
import java.util.*;

public class LowerBound {
    // Collection of Integer, Number, Object
    public static void printCollection_2(Collection<? super Integer> c) {
    }
}
```

### 9.5. Unbound Wildcards

- `E<?>`
- `List<?>`: a List of unknown type

```java
import java.util.*;

public class Unbound {
    public static void printList(List<?> list) {
        for (Object elem: list)
            System.out.println(elem + " ");
    }

    public static void main(String args[]) {
        List<Integer> listOfIntegers = Arrays.asList(1, 2, 3);
        List<String> listOfStrings = Arrays.asList("a", "b", "c");
        printList(listOfIntegers);
        printList(listOfStrings);
    }
}
```
9.6. Model View Controller

- The MVC paradigm is a way of breaking an application, into three parts:
  - the model,
  - the view, and the
  - controller.
- Input → Processing → Output
- Controller → Model → View
- Picture:

  - Model: encapsulate the logic
  - View: I/O, view
  - Controller: command center

How does the communication work?
- method calls
- events (move: model, operations, view, events)

Persistence
- Controller
  - + easy to test
  - + makes it easier to re-use the mode
  - obviously more complex
- Model - Active Record Pattern + easy
  - harder to test
  - reusing may be negatively impacted
  - obviously more complex

9.7. Observer - Observable Model

- Simply, the Observer pattern allows one object (the observer) to watch another (the subject).
- The Java programming language provides support for the Model/View/Controller architecture with two classes:
  - object that wishes to be notified when the state of another object changes
  - any object whose state may be of interest, and in whom another object may register an interest

9.8. Observer

- Interface

  void update(Observable o, Object arg)

  This method is called whenever the observed object is changed. An application calls an
  Observable object’s notifyObservers method to have all the object’s observers notified of the
  change.
9.9. Observable

From api spec:

- This class represents an observable object, or "data" in the model-view paradigm. It can be subclassed to represent an object that the application wants to have observed.

An observable object can have one or more observers. An observer may be any object that implements interface Observer. After an observable instance changes, an application calling the Observable's notifyObservers method causes all of its observers to be notified of the change by a call to their update method.

The order in which notifications will be delivered is unspecified. The default implementation provided in the Observable class will notify Observers in the order in which they registered interest, but subclasses may change this order, use no guaranteed order, deliver notifications on separate threads, or may guarantee that their subclass follows this order, as they choose.

Note that this notification mechanism is has nothing to do with threads and is completely separate from the wait and notify mechanism of class Object.

9.10. Example

```java
import java.util.Observer;
import java.util.Observable;

public class TheObserved implements Observer {
    int id = 0;
    static int counter = 0;

    public TheObserved() {
        id = counter ++;
    }

    public void update(Observable aObservable, Object o) {
        if (o instanceof String)
            System.out.println("TheObserved:update: a String object came in");
        System.out.println("TheObserved:update:o "+o);
    }
}
```

Source Code: Src/13/TheObserved.java

```java
import java.util.Observer;
import java.util.Observable;

public class TestObserved extends Observable {
    String name = null;
}
```
public TestObserved(String name) {
    this.name = name;
}

public void setName(String name) {
    this.name = name;
    setChanged();
    notifyObservers(name);
}

public void go(Integer value) {
    this.value = value;
    setChanged();
    notifyObservers(value);
}

Source Code: Src/13/TestObserved.java

public class Main {
    public static void main(String args[]) {
        TestObserved s = new TestObserved("Dr Hook and the Medicine Show");
        TheObserved aTheObserved = new TheObserved();
        s.addObserver(aTheObserved);
        s.setName("Grachmusicoff");
        s.go(1234);
    }
}

Source Code: Src/13/Main.java

Output
% java Main
TheObserved:update: a String object came in
TheObserved:update: o Grachmusicoff
TheObserved:update: ! a String object came in
TheObserved:update:o 1234

10. Exceptions and Assertions

10.1. Exceptions

See also: and

From:
When a Java program violates the semantic constraints of the Java language, a Java Virtual Machine signals this error to the program as an exception. An example of such a violation is an attempt to index outside the bounds of an array.

Java specifies that an exception will be thrown when semantic constraints are violated and will cause a non-local transfer of control from the point where the exception occurred to a point that can be specified by the programmer. An exception is said to be thrown from the point where it occurred and is said to be caught at
the point to which control is transferred.
Java programs can also throw exceptions explicitly, using throw statement.
The Java language checks, at compile time, that a Java program contains handlers for checked exceptions, by analyzing which checked exceptions can result from execution of a method or constructor.
An error indicates a serious, most likely not recoverable, situation.
10.2. Runtime Exceptions

- unchecked exceptions classes are the class RuntimeException and its subclasses
- class Error and its subclasses

10.3. Compile Time Exceptions

- All others

10.4. Runtime Exceptions are Not Checked

The runtime exception classes (RuntimeException and its subclasses) are exempted from compile-time checking because, in the judgment of the designers of Java, having to declare such exceptions would not aid significantly in establishing the correctness of Java programs. Many of the operations and constructs of the Java language can result in runtime exceptions. The information available to a Java compiler, and the level of analysis the compiler performs, are usually not sufficient to establish that such runtime exceptions cannot occur, even though this may be obvious to the Java programmer. Requiring such exception classes to be declared would simply be an irritation to Java programmers.

For example, certain code might implement a circular data structure that, by construction, can never involve null references; the programmer can then be certain that a NullPointerException cannot occur, but it would be difficult for a compiler to prove it. The theorem-proving technology that is needed to establish such global properties of data structures is beyond the scope of this Java Language Specification.
10.5. Runtime Exceptions--The Controversy

Copied from:

- Although Java requires that methods catch or specify checked exceptions, they do not have to catch or specify runtime exceptions, that is, exceptions that occur within the Java runtime system.
- Because catching or specifying an exception is extra work, programmers may be tempted to write code that throws only runtime exceptions and therefore doesn’t have to catch or specify them.
- This is "exception abuse" and is not recommended.

10.6. Throwable and Error

Please see:

- An Error is a subclass of Throwable that indicates serious problems that a reasonable application should not try to catch.
- Most such errors are abnormal conditions.
- The ThreadDeath error, though a "normal" condition, is also a subclass of Error because most applications should not try to catch it.

```java
public class ErrorE {
    private void thisMethodThrowsAnE(int index) throws Exception, Error {
        if (index == 0) {
            System.out.println("thisMethodThrowsAnException() ---> ");
            throw new Exception("in thisMethodThrowsAnException");
        } else {
            System.out.println("thisMethodThrowsAnError() ---> ");
            throw new Error("in thisMethodThrowsAnException");
        }
    }

    private void caller() {
        for (int index = 0; index < 2; index++) {
            try {
                thisMethodThrowsAnE(index);
            } catch (Exception e) {
                e.printStackTrace();
            } catch (Error e) {
                e.printStackTrace();
            } finally {
                System.out.println("Finally");
                System.out.println("Ok, a few things to clean up");
            }
        }
    }

    public static void main(String[] args) {
        new ErrorE().caller();
    }
}
```

Source Code: Src/7/ErrorE.java
java ErrorE  
thisMethodThrowsAnException() ---> 
java.lang.Exception: in thisMethodThrowsAnException  
at ErrorE.thisMethodThrowsAnE(ErrorE.java:7)  
at ErrorE.caller(ErrorE.java:18)  
at ErrorE.main(ErrorE.java:31)  

Finally  
Ok, a few things to clean up  

thisMethodThrowsAnError() --->  
java.lang.Error: in thisMethodThrowsAnException  
at ErrorE.thisMethodThrowsAnE(ErrorE.java:10)  
at ErrorE.caller(ErrorE.java:18)  
at ErrorE.main(ErrorE.java:31)  

Finally  
Ok, a few things to clean up  

10.7. Try  
When an exception is thrown, control is transferred from the code that caused the exception to the nearest  
dynamically-enclosing catch clause of a try statement that handles the exception.  
Syntax:  
try  
{  
    statement sequence  
}  

The exception can be thrown and caught in the same try block if necessary.  

try  
{  
    f();  
}  
catch (Exception e) {  
    throw e;  
}
10.8. Catch

It can be followed by zero or more catch blocks:

```
catch ( parameter )
{
    statement sequence
}
```

Example:
```
try {
    anObject.f();
    anObject.g();
} catch (SomeException_1 e) {
    // do something to recover
} catch (SomeException_2 e) {
    // do something to recover
}
```
10.9. Finally
The finally block will be always executed, regardless of what happens in the try block. (with the exception of system exit)
This provides a place where you can put statements which will be always executed.

```
finally ( )
{
    statement sequence
}
```

```java
public class Finally_1 {

    private void caller() {
        try {
            throw new Exception("in thisMethodThrowsAnException");
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            System.out.println("Finally");
        }
    }

    private void exit() {
        try {
            throw new Exception("in thisMethodThrowsAnException");
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("calling System.exit(0)");
            System.exit(0);
        } finally {
            System.out.println("Finally");
        }
    }

    public static void main(String[] args) {
        new Finally_1().caller();
        new Finally_1().exit();
    }
}
```

Source Code: Src/7/Finally_1.java

static String readFirstLineFromFileWithFinallyBlock(String path) throws IOException {
    BufferedReader br = new BufferedReader(new FileReader(path));
    try {
        return br.readLine();
    } finally {
        if (br != null) br.close();
    }
}
10.10. Throw
A throw statement allows an exception to be thrown.
Syntax:
\[
\text{throw type ThrowableException;}
\]
Example:
\[
\text{throw new Exception("Nope, this was not too good!");}
\]
10.11. Exceptions are Precise

Exceptions in Java are precise: when the transfer of control takes place, all effects of the statements executed and expressions evaluated before the point from which the exception is thrown must appear to have taken place.

No expressions, statements, or parts thereof that occur after the point from which the exception is thrown may appear to have been evaluated.

If optimized code has speculatively executed some of the expressions or statements which follow the point at which the exception occurs, such code must be prepared to hide this speculative execution from the user-visible state of the Java program.
10.12. Handling Asynchronous Exceptions

- Most exceptions occur synchronously as a result of an action by the thread in which they occur
- An asynchronous exception is an exception that can potentially occur at any point in the execution of a program.
- Asynchronous exceptions are rare. They occur only as a result of:
  - An invocation of the `stop()` (Thread or ThreadGroup)
  - An internal error in the Java virtual machine

10.13. Example 1

```java
/**
 * This class plays with exceptions
 */

public class Excep_1 {

    private int convert(String s) {
        int result = 0;
        try {
            result = Integer.parseInt(s);
        } catch (NumberFormatException e) {
            System.out.println("Haeh? "+e);
            e.printStackTrace();
        }
        return result;
    }

    public static void main(String[] args) {
        new Excep_1().convert("42");
        new Excep_1().convert("opa");
    }
}
```

Source Code: Src/7/Excep_1.java

Result:

```
% java Excep_1
Haeh? java/lang/NumberFormatException: opa
java/lang/NumberFormatException: opa
    at java/lang/Integer.parseInt(Integer.java)
    at java/lang/Integer.parseInt(Integer.java)
    at Excep_1.convert(Excep_1.java:18)
    at Excep_1.main(Excep_1.java:28)
```
10.14. Example 2

```java
public class Excep_2 {

    private void f(int n) throws NullPointerException, InterruptedException {
        System.out.println("f(\" + n + \\") ");
        switch (n) {
        case 1: throw new NullPointerException("1");
        default: throw new InterruptedException("default");
        }
    }

    public static void main(String[] args) {
        for (int index = 1; index < 3; index ++ ) {
            try {
                new Excep_2().f(index);
            } catch (NullPointerException e) {
                e.printStackTrace();
            }
            catch (Exception e) {
                System.out.println(e.getMessage() );
            }
        }
    }
}
```

Source Code: Src/7/Excep_2.java

Result:

```bash
% java Excep_2
f(1)
java/lang/NullPointerException: 1
    at Excep_2.f(Excep_2.java:17)
    at Excep_2.main(Excep_2.java:25)
f(2)
default
```

Typical compiler errors:
Exception java/lang/Exception must be caught, or it must be declared in the throws clause of this method.
    new Excep_2().f(3);

1 error
10.15. Example 3

```java
public class Excep_3 {

    private void thisMethodThrowsAnException() throws Exception {
        System.out.println("thisMethodThrowsAnException() ---> ");
        throw new Exception("in thisMethodThrowsAnException");
    }

    private void caller() {
        try {
            new Excep_3().thisMethodThrowsAnException();
            return;
        } catch (Exception e) {
            e.printStackTrace();
            return;
        } finally {
            System.out.println("Finally");
            System.out.println("Ok, a few things to clean up" );
        }
    }

    public static void main(String[] args) {
        new Excep_3().caller();
    }
}
```

Result:

```
% java Excep_3
thisMethodThrowsAnException() --->
java.lang.Exception: in thisMethodThrowsAnException
    at Excep_3.thisMethodThrowsAnException(Excep_3.java:5)
    at Excep_3.caller(Excep_3.java:18)
    at Excep_3.main(Excep_3.java:30)

Finally
Ok, a few things to clean up
```

Typical compiler errors:
Exception java/lang/Exception must be caught, or it must be declared in the throws clause of this method.

    new Excep_2().f(3);

1 error
10.16. Try Example

```java
/**
 * This class plays with exceptions
 *
 * @version $Id$
 *
 * @author hp bischof
 *
 * Revisions:
 * @Log$
 */

public class Try {

    private void f(int n) throws Exception {
        System.out.println("f(" + n + ")");
        switch (n) {
            case 1: throw new NullPointerException("1");
            default: throw new Exception("default");
        }
    }

    public static void main(String[] args) {
        int countExceptions = 0;
        for (int index = 0; index < 3; index ++ ) {
            try {
                new Try().f(index);
            } catch (Exception e) {
                e.printStackTrace();
            }
            finally {
                countExceptions ++;
            }
        }
        System.out.println("Caught " + countExceptions + " exceptions.");
    }

    Source Code: Src/7/Try.java

Result:
% java Try
f(0)
java/lang/Exception: default
    at Try.f(Try.java:18)
    at Try.main(Try.java:26)
f(1)
java/lang/NullPointerException: 1
    at Try.f(Try.java:17)
```
at Try.main(Try.java:26)

f(2)
java/lang/Exception: default
    at Try.f(Try.java:18)
    at Try.main(Try.java:26)
Caught 3 exceptions.
10.17. Example Throw and Re-throw

```java
public class Deep {
    static int exceptionCounter = 0;
    static final int MAX = 2;

    private void importantFunction(int n) throws NullPointerException, InterruptedException {
        System.out.println("importantFunction -->");
        switch (n) {
            case 1: throw new NullPointerException("1");
            default: throw new InterruptedException("default");
        }
    }

    private void smartFunction() throws Exception {
        try {
            importantFunction(exceptionCounter);
            importantFunction(exceptionCounter);
        } catch (NullPointerException e) {
            e.printStackTrace();
            throw new Exception("Programming error, Please call 555 1234 321");
        } catch (InterruptedException e) {
            e.printStackTrace();
            throw new Exception("User Error error, Please call your brain");
        } finally {
            if (++exceptionCounter >= MAX ) {
                System.err.println("Something is wrong");
                System.err.println("BYE");
                System.exit(1);///// never do this
            }
        }
    }

    public static void main(String[] args) {
        try {
            Deep aDeep = new Deep();
            System.out.println("----> ");
            aDeep.smartFunction();
            System.out.println("====> ");
            aDeep.smartFunction();
        } catch (Exception e) {
            System.out.println("Main ");
            e.printStackTrace();
        }
    }
}
```

Source Code: Src/7/Deep.java
Result:
% java Deep
----->
importantFunction -->
java.lang.InterruptedException: default
    at Deep.importantFunction(Deep.java:10)
    at Deep.smartFunction(Deep.java:17)
    at Deep.main(Deep.java:39)
Main
java.lang.Exception: User Error error, Please call your brain
    at Deep.smartFunction(Deep.java:24)
    at Deep.main(Deep.java:39)
10.18. Exceptions and Inheritance

```
   // What is the execution order?
   public class ExceptionsAndInheritance1 {
     public void importantFunction() throws InterruptedException {
       System.out.println("ExceptionsAndInheritance1:importantFunction -->");
       throw new InterruptedException("ExceptionsAndInheritance1.java");
     }

     public static void main(String[] args) {
       try {
         new ExceptionsAndInheritance1().importantFunction();
       } catch (Exception e) {
         System.out.println("Main ");
         e.printStackTrace();
       }
     }
   }

   Source Code: Src/7/ExceptionsAndInheritance1.java
```

```
   // What is the execution order?
   public class ExceptionsAndInheritance2 extends ExceptionsAndInheritance1 {
     private void importantFunction() {
       public void importantFunction() {
         System.out.println("ExceptionsAndInheritance2:importantFunction -->");
       }
     }

     public static void main(String[] args) {
       ExceptionsAndInheritance2 e2 = new ExceptionsAndInheritance2();
       ExceptionsAndInheritance1 e1 = (ExceptionsAndInheritance2)e2;
       e2.importantFunction();
       try {
         e1.importantFunction();
       } catch (Exception e) {
         System.out.println("Main ");
         e.printStackTrace();
       }
     }
   }

   Source Code: Src/7/ExceptionsAndInheritance2.java
```

Will it compile? explain your answer.
10.19. A 'Real Example'

Use:

```java
public class TestAbstract {
    public static void main(String args[])
    {
        Square aSquare;
        Circle aCircle;

        for (int index = 1; index >= -1; index -= 2 ) {
            try {
                aSquare = new Square(index);
                aCircle = new Circle(index);

                System.out.println( "Circle" );
                System.out.println( "\t" + aCircle.area() );
                System.out.println( "\t" + aCircle.perimeter() );

                System.out.println( "Square" );
                System.out.println( "\t" + aSquare.area() );
                System.out.println( "\t" + aSquare.perimeter() );
            } catch (Exception e) {
                System.out.println(e.getMessage());
            }
        }
    }
}

Source Code: Src/7/TestAbstract.java
```
abstract class Area {
    public abstract int area() throws Exception;
    public abstract int perimeter() throws Exception;
}

Source Code: Src/7/Area.java
/**
 * This class implements a Circle class.
 * @version $Id$
 * @author hp bischof
 * Revisions:
 * $Log$
 */

public class Circle extends Area {
    private int radius;

    public Circle(int _radius) throws Exception {
        if (radius < 0) {
            throw new Exception("Negativ radius (" +
                                 radius + ") is not acceptable");
        } else {
            radius = _radius;
        }
    }

    public int area() throws Exception {
        if (radius < 0) {
            throw new Exception("Circle is not initialized");
        } else {
            return (int)(Math.PI * radius * radius);
        }
    }

    public int perimeter() throws Exception {
        if (radius < 0) {
            throw new Exception("Circle is not initialized");
        } else {
            return (int)(Math.PI * radius * radius);
        }
    }
}

Source Code: Src/7/Circle.java
public class Square extends Area {

    private int length;

    public Square(int _length) throws Exception {
        if (_length < 0 )
            throw new Exception("Negative length (" +
                                 length + ") is not acceptable");
        else
            length = _length;
    }

    public int area() throws Exception {
        if ( length < 0 )
            throw new Exception("Square is not initialized");
        else
            return length * length;
    }

    public int perimeter() throws Exception {
        if ( length < 0 )
            throw new Exception("Square is not initialized");
        else
            return 4 * length;
    }

    Source Code: Src/7/Square.java
import java.io.*;
public class FileIO {

    static void cp(String inF, String outF) {
        DataInputStream in;
        DataOutputStream out;

        try {
            in = new DataInputStream(
                new FileInputStream(inF));
            out = new DataOutputStream(
                new FileOutputStream(outF));
            try {
                do {
                    out.writeByte(in.readByte());
                } while (true);
            }
            catch (EOFException e) {
                in.close();
                out.close();
            }
        }
        catch (FileNotFoundException e) {
            e.printStackTrace();
            System.out.println("Can't find the file!");
        }
        catch (IOException e) { // Throws: IOException !!!
            e.printStackTrace();
            System.out.println("Could not be opened for writing!");
        }
        catch (Exception e) {
            System.out.println("Can't find the file!");
        }
    }

    public static void main(String args[]) {
        if (args.length != 2)
            System.out.println("Usage: java FileIO f1 f2");
        else {
            cp(args[0], args[1]);
        }
    }
}

Source Code: Src/7/FileIO.java
Result:

```java
% java FileIO x xx
x x
% ls -l x
-rw------- 1 hpb fac 24 Mar 19 11:06 x
% java FileIO x xx
x xx
% chmod 000 x
% java FileIO x xx
x xx
java/gioFileNotFoundException: x
   at java/gio.FileInputStream.<init>(FileInputStream.java)
   at FileIO.cp(FileIO.java:13)
   at FileIO.main(FileIO.java:45)
Can’t find the file!
% chmod 644 x ; chmod 000 xx; ls -l x xx
-rw-r--r-- 1 hpb fac 24 Mar 19 11:06 x
-------- 1 hpb fac 24 Mar 19 11:08 xx
% java FileIO x xx
x xx
java/gioFileNotFoundException: xx
   at java/gio.FileOutputStream.<init>(FileOutputStream.java)
   at FileIO.cp(FileIO.java:15)
   at FileIO.main(FileIO.java:45)
Can’t find the file!
% rm xx
rm: xx: override protection 0 (yes/no)? y
% java FileIO x xx
x xx
java/io.FileNotFoundException: x
   at java/gio.FileInputStream.<init>(FileInputStream.java)
   at FileIO.cp(FileIO.java:13)
   at FileIO.main(FileIO.java:45)
Can’t find the file!
```
10.21. Exception Index

- ArithmeticException
- ArrayIndexOutOfBoundsException
- ArrayStoreException
- ClassCastException
- ClassNotFoundException
- CloneNotSupportedException
- Exception
- IllegalArgumentException
- IllegalArgumentException
- IllegalMonitorStateException
- IllegalStateException
- IllegalThreadStateException
- IndexOutOfBoundsException
- InstantiationException
- InterruptedException
- NegativeArraySizeException
- NoSuchFieldException
- NoSuchMethodException
- NullPointerException
- NumberFormatException
- RuntimeException
- SecurityException
- StringIndexOutOfBoundsException
- ...
10.22. Exception Class

See also

**Exception()**
Constructs an Exception with no specified detail message.

**Exception(String)**
Constructs an Exception with the specified detail message.

10.23. Throwable

Exception Class is a subclass of

- `fillInStackTrace()`: Fills in the execution stack trace.
- `getLocalizedMessage()`: Creates a localized description of this Throwable.
- `getMessage()`: Returns the detail message of this Throwable object.
- `printStackTrace()`: Prints this Throwable and its backtrace to the standard error stream.
- `printStackTrace(PrintStream)`: Prints this Throwable and its backtrace to the specified print stream.
- `printStackTrace(PrintWriter)`: Prints this Throwable and its backtrace to the specified print writer.
- `toString()`: Returns a short description of this throwable object.
10.24. Create a new Exception class

Use standard inheritance techniques. The superclass should be or a sub class.

```java
/**
 * Thrown to indicate that a method has been passed
 * an illegal or inappropriate ssid.
 */

class NumberException extends Exception {
    /**
     * Constructs a NumberException with no detail message
     */
    public NumberException() {
        super();
    }

    /**
     * Constructs a NumberException with Number.Exception detail message
     * @param s the detail message.
     */
    public NumberException(String s) {
        super(s);
    }

    Source Code: Src/7/NumberException.java
```
10.25. Student Question

```java
public class Test {
    /*
     * http://download.oracle.com/javase/tutorial/essential/exceptions/finally.html
     */
    public int tryCatchFinally() {
        try {
            try {
                System.out.println("TRY");
                int i = 1/0;
                System.out.println("TRY after 1/0");
                return 1;
            } catch (Exception e) {
                System.out.println("CATCH");
                int i = 1/0;
                System.out.println("CATCH after 1/0");
                return 2;
            } finally {
                System.out.println("FINALLY");
                int i = 1/0;
                return 3;  // what will happen if we comment this line out?
            }
        } catch (Exception e) {
            System.out.println("Y");
            return 4;
        } finally {
            System.out.println("XXX FINALLY");
            return 5;  // what will happen if we comment this line out?
        }
    }

    public static void main(String[] args) {
        // return value is?
        System.out.println("new Test().tryCatchFinally(); = " +
            new Test().tryCatchFinally());
    }
}
```

Source Code: Src/7/Test.java

- Will it compile?
- Will it execute?

10.26. Student Question II

```java
public class Test_2 {
    /*
     * http://download.oracle.com/javase/tutorial/essential/exceptions/finally.html
     */
```
public int tryCatchFinally() {
    try {
        System.out.println("TRY");
        int i = 1/0;
        System.out.println("TRY after 1/0");
        return 1;
    } catch (Exception e) {
        System.out.println("CATCH");
        try {
            int i=1 /0;
        } catch (Exception ee) {
            System.out.println("nested catch");
        }
        System.out.println("CATCH after 1/0");
        return 2;
    } finally {
        System.out.println("FINALLY");
        // return 3; // same question
    }
}

public static void main(String[] args) {
    // is the return value 2?
    System.out.println("new Test_2().tryCatchFinally(); = " +
    new Test_2().tryCatchFinally());
}

Source Code: Src/7/Test_2.java

- Will it compile?
- Will it execute?

10.27. Assertions

- assertions are used for pre/post conditions.
- An assertion is a boolean expression that a programmer specifically proclaims to be true during program runtime execution
- Declaration:
  
  assert expression1;
  assert expression1 : errorMessageExpr

10.28. Assertions: Example

    /*
     * Execution: java -ea Assertion_1
     */
    public class Assertion_1 {
        public void method( int value ) {
            assert 0 <= value;
7 System.out.println("assertM --->");
8 System.out.println("\tvalue = " + value);
9 System.out.println("assertM <----");
10 }
11
12 public static void main( String[] args ) {
13 Assertion_1 assertM = new Assertion_1();
14 assertM.method( 1 );
15 assertM.method( -1 );
16 }
17 }

Source Code: Src/7/Assertion_1.java

% javac Assertion_1.java
% java Assertion_1
assertM --->
value = 1
assertM <----
assertM --->
value = -1
assertM <----
% java -ea Assertion_1
assertM --->
value = 1
assertM <----
Exception in thread "main" java.lang.AssertionError
 at Assertion_1.method(Assertion_1.java:6)
 at Assertion_1.main(Assertion_1.java:15)

10.29. Assertions: Processed

if if true
then yes
 : No further action
else no
  if if expression2 exists
  then yes
   : Evaluate expression2 and use the result in a single-parameter form of the AssertionError constructor
   else np
    use the default AssertionError constructor

10.30. Assertions: Enable

- Java command line argument -ea
  % java -verbose
  ...
  -ea[:<packagename>...|:<classname>]
  -enableassertions[:< packagename>...|:<classname>]
   enable assertions
  -da[:<packagename>...|:<classname>]
  -disableassertions[:<packagename>...|:<classname>]
   disable assertions
-esa | -enablesystemassertions
    enable system assertions
-dsa | -disablesystemassertions
    disable system assertions
...

10.31. Assertions: Throwable

See also:

- Source:

```java
/*
 * Execution: java -ea:
 */

public class Assertion_2 {
    public void method( int value ) {
        assert 0 <= value: "Value must be positive =" + value + ":
        System.out.println("assertM ---->
        System.out.println("\tvalue = " + value
        System.out.println("assertM <----
    }

    public static void printAssertion( AssertionError ae ) {
        StackTraceElement[] stackTraceElements = ae.getStackTrace();
        StackTraceElement stackTraceElement = stackTraceElements[ 0 ];
        System.err.println( "AssertionError" );
        System.err.println( "class= " + stackTraceElement.getClassName() );
        System.err.println( "method= " + stackTraceElement.getMethodName() );
        System.err.println( "message= " + ae.getMessage() );
    }

    public static void main( String[] args ) {
        Assertion_2 assertM = new Assertion_2();
        try {
            assertM.method( 1 );
            assertM.method( -1 );
        } catch( AssertionError ae ) {
            printAssertion(ae);
        }
    }
}

Source Code: Src/7/Assertion_2.java
```

```sh
% java -ea Assertion_2
assertM ---->
    value = 1
assertM <----
AssertionError
    class= Assertion_2
    method= method
    message= Value must be positive =-1=
```
10.32. Assertions: Disabling

- Assertions can also be disabled down to the class level
- the command line argument `-disableassertions` (-da parallels the syntax of the assertion-enabling switch.
- a command line can contain as many enable- and disable-assertion switches as desired.
11. I/O: Files and Streams

See also:

File:

- Files can store persistent information.
- Objects can be stored in a file.
- Files offer random access.
- Files can be created, removed, overwritten and appended.
- Files must have a name — the name depends on the operating system. They don’t depend on Java.

In Java, file I/O as well as keyboard/screen I/O is handled by streams.
11.1. Overview

- Package has two inheritance chains that can be used for dealing with files. One chain starts with the abstract classes and and the second chain starts with and
  
  InputStream and OutputStream and their subclasses are used for input and output of byte values.

  InputStreamReader and OutputStreamWriter combine an encoding and a stream and thus are the preferred classes for text input and output. This means they are byte based.

- The Reader and Writer classes are defined for reading from character files. They provide facilities specific to character files, such as, reading a line of input.

  Reader and Writer classes are abstract — they can have abstract methods (declaration without a body) and no objects may be instantiated. Abstract classes define basic behavior for related subclasses including instance variables and some method implementations.

  Text files normally contain byte values that represent a selection of char values by way of an encoding like 8859_1 (ISO Latin-1), or a code page like Cp850 (PC Latin-1), or with the unicode transfer format UTF8.
11.2. Input Stream

For an input stream, the source of data might be a file, a String, an array of bytes, or bytes written to an output stream (typically by another thread). There are also "filter input streams" that take data from another input stream and transform or augment the data before delivering it as input. For example, a passes bytes through verbatim but counts line terminators as they are read.
The drawings have been created by

11.3. Output Stream

For an output stream, the sink of data might be a file, an array of bytes, or a buffer to be read as an input stream (typically by another thread). There are also "filter output streams" that transform or augment data before writing it to some other output stream.

An instance of class File represents a path name (a String) that might identify a particular file within a file system. Certain operations on the file system, such as renaming and deleting files, are done by this class rather than through streams.
11.4. Using Streams

No matter where the information is coming from or going to the algorithm for reading/writing is pretty much always the same:

Reading:

open a stream for reading
while more information
    read
    process

close stream

Writing:

open a stream for writing
while more information
    process
    write

close stream
11.5. File Descriptor

An instance of class represents an abstract indication of a particular file within a file system; such file descriptors are created internally by the Java I/O system.

Methods:

public native boolean valid()

Tests if this file descriptor object is valid. Returns: true if the file descriptor object represents a valid, open file or socket; false otherwise.

sync

Force all system buffers to synchronize with the underlying device. This method returns after all modified data and attributes of this FileDescriptor have been written to the relevant device(s). In particular, if this FileDescriptor refers to a physical storage medium, such as a file in a file system, sync will not return until all in-memory modified copies of buffers associated with this FileDescriptor have been written to the physical medium.

The Solaris man

sync() causes all information in memory that should be on disk to be written out. This includes modified super blocks, modified i-nodes, and delayed block I/O.

It should be used by programs that examine a file system, such as fsck.1m df.1m etc. It is mandatory before a re-boot.

The writing, although scheduled, is not necessarily completed before sync() returns. The fsync function completes the writing before it returns.
### 11.6. Data Processing Streams

Processing streams perform some sort of operation, such as buffering or character encoding, as they read and write. Like the data sink streams, java/io often contains pairs of streams: one that performs a particular operation during reading and another that performs the same operation (or reverses it) during writing. This table gives java/io’s processing streams.

<table>
<thead>
<tr>
<th>Process</th>
<th>Character Stream</th>
<th>Byte Stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffering</td>
<td>BufferedReader,</td>
<td>BufferedInputStream,</td>
</tr>
<tr>
<td></td>
<td>BufferedWriter</td>
<td>BufferedWriterOutputStream</td>
</tr>
<tr>
<td>Filtering</td>
<td>FilterReader,</td>
<td>FilterInputStream,</td>
</tr>
<tr>
<td></td>
<td>FilterWriter</td>
<td>FilterOutputStream</td>
</tr>
<tr>
<td>Converting between</td>
<td>InputStreamReader,</td>
<td></td>
</tr>
<tr>
<td>Bytes and Characters</td>
<td>OutputStreamReader,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OutputStreamWriter</td>
<td></td>
</tr>
<tr>
<td>Concatenation</td>
<td></td>
<td>SequenceInputStream</td>
</tr>
<tr>
<td>Object Serialization</td>
<td></td>
<td>ObjectInputStream,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ObjectOutputStream</td>
</tr>
<tr>
<td>Data Conversion</td>
<td></td>
<td>DataInputStream,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DataOutputStream</td>
</tr>
<tr>
<td>Counting</td>
<td>LineNumberReader</td>
<td>LineNumberInputStream</td>
</tr>
<tr>
<td>Peeking Ahead</td>
<td>PushbackReader</td>
<td>PushbackInputStream</td>
</tr>
<tr>
<td>Printing</td>
<td>PrintWriter</td>
<td>PrintStream</td>
</tr>
</tbody>
</table>
11.7. Data Sink Streams

Data sink streams read from or write to specialized data sinks such as strings, files, or pipes. Typically, for each reader or input stream intended to read from a specific kind of input source, java/io contains a parallel writer or output stream that can create it. The following table gives java/io’s data sink streams.

<table>
<thead>
<tr>
<th>Sink Type</th>
<th>Character Streams</th>
<th>Byte Streams</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>CharArrayReader,</td>
<td>ByteArrayOutputStream,</td>
</tr>
<tr>
<td></td>
<td>CharArrayWriter</td>
<td>ByteArrayInputStream,</td>
</tr>
<tr>
<td></td>
<td>StringWriter</td>
<td>StringBufferInputStream,</td>
</tr>
<tr>
<td></td>
<td>StringReader</td>
<td>StringBufferOutputStream</td>
</tr>
<tr>
<td>Pipe</td>
<td>PipedReader,</td>
<td>PipedInputStream,</td>
</tr>
<tr>
<td></td>
<td>PipedWriter</td>
<td>PipedOutputStream,</td>
</tr>
<tr>
<td>File</td>
<td>FileReader,</td>
<td>FileOutputStream,</td>
</tr>
<tr>
<td></td>
<td>FileWriter</td>
<td>FileOutputSteam</td>
</tr>
</tbody>
</table>

CharArrayReader and CharArrayWriter
ByteArrayInputStream and ByteArrayInputStream

Use these streams to read from and write to memory. You create these streams on an existing array and then use the read and write methods to read from or write to the array.

FileReader and FileWriter
FileStream and FileOutputStream

Collectively called file streams, these streams are used to read from or write to a file on the native file system.

PipedReader and PipedWriter
PipedInputStream and PipedOutputStream

Implement the input and output components of a pipe. Pipes are used to channel the output from one program (or thread) into the input of another.

StringReader and StringWriter
StringBufferInputStream

Use StringReader to read characters from a String as it lives in memory. Use StringWriter to write to a String. StringWriter collects the characters written to it in a StringBuffer, which can then be converted to a String. StringBufferInputStream is similar to StringReader, except that it reads bytes from a StringBuffer.
11.8. StreamTokenizer

The class provides some simple support for parsing bytes or characters from an input stream into tokens such as identifiers, numbers, and strings, optionally ignoring comments and optionally recognizing or ignoring line terminators.

Java/Io

The hierarchy of classes defined in package
11.9. A Copy Program

```java
import java.io.*;

public class InOut_1 {
    public static void main( String args[] ) {
        DataInputStream in;
        DataOutputStream out;
        byte[] buffer = new byte[1024];
        int n;

        if ( args.length < 2 ) {
            System.err.println(
                "Usage: java InOut_1 from to");
            System.exit(1);
        }

        try {
            in = new DataInputStream(
                new FileInputStream(args[0]) );
            out = new DataOutputStream(
                new FileOutputStream(args[1]) );

            while ( (n = in.read(buffer) ) != -1 ) {
                out.write(buffer, 0, n);
            }

            out.close();
            in.close();

        } catch ( FileNotFoundException ef) {
            System.out.println("File not found: " + args[1]);
        }
        catch ( IOException ef) {
            System.out.println("File not found: " + args[1]);
        }
        catch ( Exception e) {
            System.out.println("ExceptionType occurred: " +
                e.getMessage() );
        }
    }
}

Source Code: Src/9_was/InOut_1.java
```
Result:

% java InOut_1
% java InOut_1 InOut_2.class x
% diff InOut_1.class x
%
11.10. Size Matters: The second Copy Program

Does the size of the read and write blocks matter?

```java
import java.io.*;

public class InOut_2 {
    static final int BUFSIZE = 1024;
    public static void copy( String inF, String outF, int bufSize ) {
        DataInputStream in;
        DataOutputStream out;
        byte[] buffer = new byte[bufSize];
        int n;

        try {
            in = new DataInputStream( new FileInputStream(inF) );
            out = new DataOutputStream( new FileOutputStream(outF) );
            while ( (n = in.read(buffer) ) != -1 ) {
                out.write(buffer, 0, n);
            }
            out.close();
            in.close();
        } catch (FileNotFoundException ef) {
            System.out.println(ef.getMessage() );
        } catch (IOException ef) {
            System.out.println(ef.getMessage() );
        } catch (Exception e) {
            System.out.println("ExceptionType occurred: " + e.getMessage() );
        }

        public static void main( String args[] ) {
            int bufSize = BUFSIZE;

            if ( args.length < 2 ) {
                System.err.println("Usage: java InOut_1 from to [size]");
                System.exit(1);
            }

            if ( args.length == 3 ) {
```
try {
   ,bufSize = Integer.parseInt(args[2]);
} 
catch (NumberFormatException e) {
    System.out.println("Can’t convert " + args[2] 
 + " to an integer.");
}

System.out.println("BufferSize = " + bufSize);
copy(args[0], args[1], bufSize);
<table>
<thead>
<tr>
<th>BufferSize</th>
<th>real</th>
<th>user</th>
<th>sys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>49.8</td>
<td>21.3</td>
<td>24.4</td>
</tr>
<tr>
<td>2</td>
<td>27.0</td>
<td>10.8</td>
<td>12.3</td>
</tr>
<tr>
<td>512</td>
<td>0.8</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>1024</td>
<td>0.8</td>
<td>0.1</td>
<td>0.2</td>
</tr>
<tr>
<td>10240</td>
<td>1.0</td>
<td>0.2</td>
<td>0.1</td>
</tr>
</tbody>
</table>
11.11. Reading from Stdin

```java
import java.io.*;

public class stdin {
    public static void main( String args[] ) {
        LineNumberInputStream input;

        if ( args.length > 1 ) {
            System.err.println("Usage: java stdin file-name");
            System.exit(1);
        }

        try {
            String line;
            //
            if ( args.length == 1 )
                input = new LineNumberInputStream(
                    new DataInputStream(
                        new FileInputStream(args[0]) ) );
            else
                input = new LineNumberInputStream( System.in );

            while ( ( input.read() ) != -1 ) {
                System.out.println("# lines = " + input.getLineNumber() );
                input.close();
            }
            catch ( FileNotFoundException e ) {
                System.out.println(e.getMessage());
            }
            catch ( IOException e ) {
                System.out.println(e.getMessage());
            }
            catch ( Exception e ) {
                System.out.println("ExceptionType occurred: " +
                    e.getMessage() );
            }
        }
    }
}
```

Source Code: Src/14/stdin.java

11.12. Reading/Writing Compressed Files
/**
 * Opens the file with the data.
 * A rewind will happen, if this method gets called more than once.
 **/
 public void openFileForReading() {
     try {
         if (isBinaryInput) {
             bInputStream = new DataInputStream(
                 new GZIPInputStream(
                     new FileInputStream(inputFileName)
                 )
             );
         } else {
             inputStream = new BufferedReader(new FileReader(inputFileName));
         }
      ...
      } catch (Exception e) {
         e.printStackTrace();
         InOutErr.out.println("ParticleViewExtractor: openFileForReading-" + e);
     }
 }

/* Convert Data to Binary format */
public void doConvertToBinary(String fileName) {
    try {
        BufferedReader inputStream = new BufferedReader(new FileReader(fileName));
        DataOutputStream outputStream = new DataOutputStream(
            new GZIPOutputStream(
                new FileOutputStream(fileName + ...
            )
        );
    }
}

Source Code: Src/14/Compressed.java

11.13. A Grep Program
Extract from:
BufferedReader(Reader)
    Create a buffering character-input stream that uses a default-sized input buffer.
BufferedReader(Reader, int)
    Create a buffering character-input stream that uses an input buffer of the specified size.
import java.io.*;

public class Grep {

    public static void main( String args[] ) {
        BufferedReader input;
        PrintWriter output;

        if ( args.length < 2 ) {
            System.err.println("Usage: java Grep search-string file-name [output-filename]");
            System.exit(1);
        }

        try {
            String line;
            input = new BufferedReader(new FileReader(args[1]));
            if ( args.length == 3 ) {
                output = new PrintWriter(new FileWriter(args[2]));
            } else
                output = new PrintWriter(System.out);

            while ( (line = input.readLine()) != null ) {
                if (line.indexOf(args[0]) >= 0 )
                    output.println(line);
            }
            output.close();
            input.close();
        }

        catch (FileNotFoundException e) {
            System.out.println(e.getMessage());
        }
        catch (IOException e) {
            System.out.println(e.getMessage());
        }
        catch (Exception e) {
            System.out.println("ExceptionType occurred: " +
                e.getMessage());
        }
    }
}

Source Code: Src/9_was/Grep.java

% java Grep Grep Grep.java
public class Grep {
    "Usage: java Grep search-string file-name [output-filename]";
% java Grep Grep
Usage: java Grep search-string file-name [output-filename]
If you are interested in line numbers, choose LineNumberReader

```java
import java.io.*;

public class Grep2 {
    public static void main(String args[]) {
        LineNumberReader input;
        PrintWriter output;

        if (args.length < 2) {
            System.err.println("Usage: java Grep search-string file-name [outputfilename]" Kap 17
            System.exit(1);
        }

        try {
            String line;
            input = new LineNumberReader(
                new BufferedReader(
                    new FileReader(args[1])
                )
            );

            if (args.length == 3) {
                output = new PrintWriter(new FileWriter(args[2]));
            } else
                output = new PrintWriter(System.out);

            while ((line = input.readLine()) != null) {
                if (line.indexOf(args[0]) >= 0)
                    output.println(input.getLineNumber() + ": " + line);
            }
            output.close();
            input.close();
        }
        catch (FileNotFoundException e) {
            System.out.println(e.getMessage());
        }
        catch (IOException e) {
            System.out.println(e.getMessage());
        }
        catch (Exception e) {
            System.out.println("ExceptionType occurred: " + e.getMessage());
        }
    }
}
```

Source Code: Src/9_was/Grep2.java
% java Grep2 Grep Grep.java
3: public class Grep {
%
-229-

11.14. Regular Expressions
•

See also here:

11.15. Regular Expressions in Java
See also here:
•

Classes for matching character sequences against patterns specified by regular expressions.

•

An instance of the Pattern class represents a regular expression that is specified in string form in a
syntax similar to that used by Perl.

•

Instances of the Matcher class are used to match character sequences against a given pattern.

11.16. Example 1

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39

/*
* Checks for invalid characters
* in email addresses
*/
import java.util.regex.*;
public class EmailValidation {
/*
* Checks for email addresses starting with
* inappropriate symbols like dots or @ signs.
*/
public static void checkForPorA(String aPossibleEmail )
{
Pattern p = Pattern.compile("ˆ\\.|ˆ\\@");
Matcher m = p.matcher(aPossibleEmail);
if (m.find())
System.err.println(aPossibleEmail + " - Email addresses don’t start" +
" with dots or @ signs.");
else
System.err.println(aPossibleEmail + " is valid.");
}
/*
* Checks for email addresses starting with
* www.
*/
public static void checkForWWW(String aPossibleEmail )
{
Pattern p = Pattern.compile("ˆwww\\.");
Matcher m = p.matcher(aPossibleEmail);
if (m.find())
System.err.println(aPossibleEmail + " - Email addresses don’t start" +
" with www.");
else
System.err.println(aPossibleEmail + " is valid.");
}

/*


/* Checks for invalid characters in email addresses. */
public static void checkForInvalidC(String aPossibleEmail) {
    Pattern p = Pattern.compile("[^A-Za-z0-9\./@_\-\^\#]+");
    Matcher m = p.matcher(aPossibleEmail);
    StringBuffer sb = new StringBuffer();
    boolean result = m.find();
    boolean deletedIllegalChars = false;
    while(result) {
        deletedIllegalChars = true;
        m.appendReplacement(sb, "");
        result = m.find();
    }
    if (deletedIllegalChars) {
        System.out.println("It contained incorrect characters such as spaces or commas.");
    }
}

public static void main(String[] args) throws Exception {
    checkForPorA("hp@cs.rit.edu");
    checkForPorA("@cs.rit.edu");
    checkForWWW("www.cs.rit.edu");
    checkForInvalidC("hp@cs.rit.edu");
    checkForInvalidC("p b@cs.rit.edu");
}

Source Code: Src/9_reg_ex/EmailValidation.java

11.17. Example 2

/*
 * Checks for invalid characters in email addresses
 */
import java.util.regex.*;
public class TheN {
    /* Palindroms */
    public static void checkForP(String aPossibleEmail) {

Pattern p = Pattern.compile("^.$");
Matcher m = p.matcher(aPossibleEmail);
if (m.find())
    System.err.println(aPossibleEmail + " one character");
else
    System.err.println(aPossibleEmail + " more than one character");
}

public static void checkForP2(String aPossibleEmail) {
    Pattern p = Pattern.compile("^(.).\1$" );
    Matcher m = p.matcher(aPossibleEmail);
    if (m.find())
        System.err.println(aPossibleEmail + " 2 char palindrom");
    else
        System.err.println(aPossibleEmail + " ! a 2 char palindrom");
}

public static void main(String[] args) throws Exception {
    checkForP("a");
    checkForP("aa");
    checkForP2("a");
    checkForP2("ata");
    if ( Pattern.matches("^(.).\1$", "aa" ) )
        System.err.println("palindrom");
}

Source Code: Src/9_reg_ex/TheN.java

11.18. Serializing Objects

- Two of the byte streams, and are specialized streams that let you read and write objects. Reading and writing objects is a process known as object serialization. Object serialization has many uses, including remote method invocation (RMI). In addition to the object streams, java/io has other classes and interfaces that define the API to help classes perform serialization for its instances.
- Only objects that support the interface can be written to streams. The class of each serializable object is encoded including the class name and signature of the class, the values of the object's fields and arrays, and the closure of any other objects referenced from the initial objects.
- Reconstructing an object from a stream requires that the object first be written to a stream.
- The default serialization mechanism for an object writes the class of the object, the class signature, and the values of all non-transient and non-static fields. References to other objects (except in transient or static fields) cause those objects to be written also. Multiple references to a single object are encoded using a reference sharing mechanism so that graphs of objects can be restored to the same shape as when the original was written.
Object writer:

```java
import java.io.*;
import java.util.Date;

public class ObjectWriter_1 {
    public static void main( String args[] ) {
        Date d = new Date();

        try {
            FileOutputStream ostream =
                new FileOutputStream("object_1.data");
            ObjectOutputStream p = new ObjectOutputStream(ostream);
            p.writeInt(12345);
            p.writeObject("Today");
            System.out.println("String =" + "Today");
            p.writeObject(d);
            System.out.println("Date =" + d);
            p.flush();
            p.close();
        }
        catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Source Code: Src/9_was/ObjectWriter_1.java

% java ObjectWriter_1
Integer = 1234
String = Today
Date = Mon Nov 1 08:42:38 EDT 2010
import java.io.*;
import java.util.Date;

public class ObjectReader_1 {
    public static void main( String args[] ) {
        try {
            FileInputStream istream =
                new FileInputStream("object_1.data");
            ObjectInputStream p = new ObjectInputStream(istream);
            int i = p.readInt();
            System.out.println("Integer = " + i);
            String today = (String)p.readObject();
            System.out.println("String = " + today);
            Date date = (Date)p.readObject();
            System.out.println("Date = " + date);
            p.close();
            istream.close();
        }
        catch (IOException e) {
            System.out.println(e.getMessage());
        }
        catch (ClassNotFoundException e) {
            System.out.println(e.getMessage());
        }
    }
}

Source Code: Src/9_was/ObjectReader_1.java

% java ObjectReader_1
Integer = 1234
String = Today
Date = Mon Nov 1 08:42:38 EDT 2010
ls -l o*a
-rw------- 1 hpb fac 60 Oct 4 10:49 object_1.data
yps 9 85 od -c object_1.data
0000000 254 355 \0 005 w 004 \0 \0 0 9 t \0 005 T o d
0000020 a y s r \0 016 j a v a . u t i l .
0000040 D a t e h j 201 001 K Y t 031 003 \0 \0 x
0000060 p w \b \0 \0 \0 326 365 < o 232 x
0000074
When an object is serialized, any object reference it contains are also serialized. A example.

```java
import java.io.*;
import java.util.*;

public class ObjectWriter_2 {
    public static void main( String args[] ) {
       Hashtable aHashTable = new Hashtable();
aHashTable.put("plus Movie", "A little Voice");
aHashTable.put("minus Movie", "Independence Day");

        try {
            FileOutputStream ostream =
                new FileOutputStream("object_2.data");
            ObjectOutputStream p = new ObjectOutputStream(ostream);
p.writeObject(aHashTable);
            System.out.println("aHashTable = "+ aHashTable.toString());
p.flush();
p.close();
        }
        catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

% java ObjectWriter_2
aHashTable = {minus Movie=Independence Day, plus Movie=A little Voice}
import java.io.*;
import java.util.*;

public class ObjectReader_2 {
    public static void main( String args[] ) {

        Hashtable aHashTable;

        try {
            FileInputStream istream =
                    new FileInputStream("object_2.data");
            ObjectInputStream p = new ObjectInputStream(istream);

            aHashTable= (Hashtable)p.readObject();
            System.out.println("aHashTable = " + aHashTable.toString());
            p.close();
        }
        catch (IOException e) {
            System.out.println(e.getMessage());
        }
        catch (ClassNotFoundException e) {
            System.out.println(e.getMessage());
        }
    }
}

Source Code: Src/9_was/ObjectReader_2.java
% java ObjectReader_2
aHashTable = {minus Movie=Independence Day, plus Movie=A little Voice}
11.19. Can an Object include itself?

```java
import java.io.*;
import java.util.*;

public class Self {
    public static void main( String args[] ) {
        Hashtable aHashTable = new Hashtable();
        aHashTable.put("plus Movie", "A little Voice");
        aHashTable.put("The HashTable", aHashTable);

        try {
            FileOutputStream ostream =
                new FileOutputStream("self.data");
            ObjectOutputStream p =
                new ObjectOutputStream(ostream);
            p.writeObject(aHashTable);
            p.flush();
            p.close();
        }
        catch (IOException e) {
            System.out.println(e.getMessage());
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

Source Code: Src/9_was/Self.java

% java Self
% od -c self.data
0000000 254 355 \0 005 s r \0 023 j a v a . u t i
0000020 l . H a s h t a b l e 023 273 017 % !
0000040 J 344 270 003 \0 002 F \0 \n l o a d F a c ...
0000160 M o v i e t \0 016 A l i t t l
0000200 e V o i c e x
0000210
import java.io.*;
import java.util.*;

public class Self_Reader {
    public static void main( String args[] ) {
        Hashtable aHashTable;

        try {
            FileInputStream istream =
                new FileInputStream("self.data");
            ObjectInputStream p = new ObjectInputStream(istream);

            aHashTable= (Hashtable)p.readObject();
            System.out.println("plus Movie = " + aHashTable.get("plus Movie"));
            System.out.println("The HashTable" + aHashTable.get("The HashTable"));
            System.out.println("aHashTable ="+aHashTable.toString());
            p.close();
        }
        catch ( IOException e) {
            System.out.println(e.getMessage());
        }
        catch ( ClassNotFoundException e) {
            System.out.println(e.getMessage());
        }
    }

    Source Code: Src/9_was/Self_Reader.java

    % java Self_Reader 2>&1 | more // version jdk1.3.1
    plus Movie = A little Voice
    java/lang/StackOverflowError
        at java/lang/StringBuffer.<init>(StringBuffer.java)
        at java/gutil.Hashtable.toString(Hashtable.java)
        at java/gutil.Hashtable.toString(Hashtable.java)
    ...

    % java Self_Reader // version jdk1.4
    plus Movie = A little Voice
    The HashTable{The HashTable=(this Map), plus Movie=A little Voice}
aHashTable = {The HashTable=(this Map), plus Movie=A little Voice}
11.20. Make it Serializable

The first shot is using a 'Serializable' class:

```java
import java.io.*;
import java.util.*;

public class ObjectWriter_3 extends Hashtable {
    int local = 42;

    public static void main( String args[] ) {
        ObjectWriter_3 aObjectWriter_3 = new ObjectWriter_3();

        try {
            FileOutputStream ostream =
                new FileOutputStream("object_3.data");
            ObjectOutputStream p = new ObjectOutputStream(ostream);
            p.writeObject(aObjectWriter_3);
            System.out.println("aObjectWriter_3 ="+aObjectWriter_3.toString());
            System.out.println("aObjectWriter_3.local =" + aObjectWriter_3.local);
            p.flush();
            p.close();
        } catch (IOException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Source Code: Src/9_was/ObjectWriter_3.java

% java ObjectWriter_3
aObjectWriter_3 = {}
aObjectWriter_3.local = 42
import java.io.*;
import java.util.*;

public class ObjectReader_3 {
    public static void main( String args[] ) {

        ObjectWriter_3 aObjectWriter_3;

        try {
            FileInputStream istream =
                new FileInputStream("object_3.data");
            ObjectInputStream p = new ObjectInputStream(istream);

            aObjectWriter_3= (ObjectWriter_3)p.readObject();
            System.out.println("ObjectWriter_3.local = " + aObjectWriter_3.local);
            p.close();
        }
        catch ( IOException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
        catch ( ClassNotFoundException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
}

Source Code: Src/9_was/ObjectReader_3.java

% java ObjectReader
aObjectWriter_3.local = 42
An object is serializable only if its class implements the Serializable interface. Thus, if you want to serialize the instances of one of your classes, the class must implement the Serializable interface. The good news is that Serializable is an empty interface. That is, it doesn’t contain any method declarations; it’s purpose is simply to identify classes whose objects are serializable.

You don’t have to write any methods. The serialization of instances of this class are handled by the defaultWriteObject method of ObjectOutputStream. This method automatically writes out everything required to reconstruct an instance of the class, including the following:

— Class of the object
— Class signature
— Values of all non-transient and non-static members, including members that refer to other objects
```java
import java.io.*;
import java.util.*;

public class ObjectWriter_4 implements Serializable {
    int local = 42;
    
    private void writeObject(ObjectOutputStream s) throws IOException {
        s.defaultWriteObject();
        // customized serialization code
    }

    private void readObject(ObjectInputStream s) throws IOException {
        try {
            s.defaultReadObject();
        }
        catch (ClassNotFoundException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
        // customized deserialization code
        // ...
        // followed by code to update the object, if necessary
    }

    public static void main( String args[] ) {
        ObjectWriter_4 aObjectWriter_4 = new ObjectWriter_4();

        try {
            FileOutputStream ostream =
                new FileOutputStream("object_4.data");
            ObjectOutput p = new ObjectOutputStream(ostream);
            p.writeObject(aObjectWriter_4);
            System.out.println("aObjectWriter_4 =" + aObjectWriter_4.toString());
            System.out.println("aObjectWriter_4.local =" + aObjectWriter_4.local);
            p.flush();
            p.close();
        }
        catch (IOException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Source Code: Src/9_was/ObjectWriter_4.java

% java ObjectWriter_4
aObjectWriter_4 = ObjectWriter_4@1dc60810
aObjectWriter_4.local = 42

The reader program must not be modified.
11.21. StreamTokenizer

The StreamTokenizer class takes an input stream and parses it into "tokens", allowing the tokens to be read one at a time. The parsing process is controlled by a table and a number of flags that can be set to various states. The stream tokenizer can recognize identifiers, numbers, quoted strings, and various comment styles. Each byte read from the input stream is regarded as a character in the range `\u0000` through `\u00FF`. The character value is used to look up five possible attributes of the character: white space, alphabetic, numeric, string quote, and comment character. Each character can have zero or more of these attributes.

In addition, an instance has four flags. These flags indicate:

- Whether line terminators are to be returned as tokens or treated as white space that merely separates tokens.
- Whether C-style comments are to be recognized and skipped.
- Whether C++-style comments are to be recognized and skipped.
- Whether the characters of identifiers are converted to lowercase.

A typical application first constructs an instance of this class, sets up the syntax tables, and then repeatedly loops calling the nextToken method in each iteration of the loop until it returns the value TT_EOF.
The first program:

```java
import java.io.*;
public class St_1 {
    public static void main( String args[] ) {
        StreamTokenizer input;
        if ( args.length > 1 ) {
            System.err.println("Usage: java St [file-name]");
            System.exit(1);
        }
        try {
            String line;
            if ( args.length == 1 )
                input = new StreamTokenizer( new FileReader(args[0]) );
            else
                input = new StreamTokenizer( new InputStreamReader(System.in) );
            while ( input.TT_EOF != input.nextToken() ) {
                System.out.println(input.lineno() + " : " + input.toString());
            }
        }
        catch ( FileNotFoundException e ) {
            System.out.println(e.getMessage());
        }
        catch ( IOException e ) {
            System.out.println(e.getMessage());
        }
        catch ( Exception e ) {
            System.out.println("Exception occurred: " + e.getMessage() );
            e.printStackTrace();
        }
    }
}
```

Source Code: Src/9_was/St_1.java
Result:

```bash
% head -1 /etc/passwd
root:x:0:1:Super-User:/:/sbin/sh
% head -1 /etc/passwd | java St_1
1: Token[root], line 1
1: Token['], line 1
1: Token[x], line 1
1: Token['], line 1
1: Token[x], line 1
1: Token['], line 1
1: Token[n=0.0], line 1
1: Token['], line 1
1: Token[n=1.0], line 1
1: Token['], line 1
1: Token[Super-User], line 1
1: Token['], line 1

% java St_1 /etc/passwd | sed 7q
1: Token[root], line 1
1: Token['], line 1
1: Token[x], line 1
1: Token['], line 1
1: Token[n=0.0], line 1
1: Token['], line 1
1: Token[n=1.0], line 1
1: Token['], line 1

% java St_1
hello
1: Token[hello], line 1
a b:c d;e
2: Token[a], line 2
2: Token[b], line 2
2: Token['], line 2
2: Token[c], line 2
2: Token[d], line 2
2: Token['], line 2
2: Token[e], line 2

% java St_1
aa ///
1: Token[aa], line 1
sss /// www
2: Token[sss], line 2
222 + #
3: Token[n=222.0], line 3
3: Token['], line 3
3: Token[ '#'], line 3
```
The second program is a beginning of a calculator:

```java
import java.io.*;

public class St_2 {
    StreamTokenizer input;

    public void adjustT() {
        input.resetSyntax();
        input.commentChar('#'); // comments from #
        input.wordChars('0', '9'); // parse decimal
        input.wordChars('.', '.'); // operators as words
        input.wordChars('+', '+'); // operators as words
        input.wordChars('-', '-'); // operators as words
        input.wordChars('}', '}'); // operators as words
        input.whitespaceChars(0, ''); // ignore white space
        input.eolIsSignificant(true); // need '\n'
    }

    public void processInput() throws IOException {
        while (input.TT_EOF != input.nextToken()) {
            if (input.ttype != input.TT_EOL)
                System.out.println(input.lineno() + " : " + input.sval);
            else
                System.out.println("Saw EOL");
        }
    }

    public static void main( String args[] ) {
        St_2 aSt_2 = new St_2();
        if (args.length > 1) {
            System.err.println("Usage: java St [file-name]");
            System.exit(1);
        }
        try {
            String line;
            if (args.length == 1)
                aSt_2.input = new StreamTokenizer(
                    new FileReader(args[0]));
            else
                aSt_2.input = new StreamTokenizer(
                    new InputStreamReader(System.in));
            aSt_2.adjustT();
        }
```
aSt_2.processInput();

} catch (FileNotFoundException e) {
    System.out.println(e.getMessage());
}
catch (IOException e) {
    System.out.println(e.getMessage());
}
catch (Exception e) {
    System.out.println("ExceptionType occurred: "+ e.getMessage());
    e.printStackTrace();
}

Source Code: Src/9_was/St_2.java
Result:
% java St_2
Saw EOL
2 + 3
1: 2
1: +
1: 3
Saw EOL
2 - 3
2: 2
2: -
2: 3
Saw EOL
A simple calculator:

The first idea for the process loop:

```java
public void processInput() throws IOException {
    while ( input.TT_EOF != new Expression(input) ) {
        
    }
}
```

What do you think?

- Scanner:

```java
import java.io.BufferedReader;
import java.io.FilterReader;
import java.io.IOException;
import java.io.Reader;
import java.io.StreamTokenizer;

/**
   \* lexical analyzer for arithmetic expressions.
   \* Comments extend from # to end of line.
   \* Words are composed of digits and decimal point(s).
   \* White space consists of control characters and space and is ignored;
   \* however, end of line is returned.
   \* Fixes the lookahead problem for TT_EOL.
   */

public class Scanner extends StreamTokenizer {
    /** kludge: pushes an anonymous Reader which inserts 
        a space after each newline. 
        */
    public Scanner (Reader r) {
        super (new FilterReader(new BufferedReader(r)) {
            protected boolean addSpace; // kludge to add space after \n
            public int read () throws IOException {
                int ch = addSpace ? ' ' : in.read();
                addSpace = ch == '\n';
                return ch;
            }
        });
        resetSyntax();
        commentChar('#'); // comments from # to end-of-line
        wordChars('0', '9'); // parse decimal numbers as words
        wordChars('.','.');
        whitespaceChars(0,' '); // ignore control-* and space
        eolIsSignificant(true); // need '\n'
    }
}
```

Source Code: Src/9_e/Scanner.java
• Expression.java

```java
    /*
     * Thanks to ats
     */
    import java.io.InputStreamReader;
    import java.io.IOException;
    import java.io.ObjectOutputStream;
    import java.io.StreamTokenizer;
    import java.util.Vector;

    /** recognizes, stores, and evaluates arithmetic expressions.
     */
    public abstract class Expression {
        final static int eol = StreamTokenizer.TT_EOL; // switch use ...
        final static int eof = StreamTokenizer.TT_EOF; // must be const :(;
        final static int word = StreamTokenizer.TT_WORD;

        /** reads lines from standard input, parses, and evaluates them
         or writes them as a Vector to standard output if -c is set.
         @param args if -c is specified, a Vector is written.
         */
        public static void main (String args []) {
            Scanner scanner = new Scanner(new InputStreamReader(System.in));
            try {
                do
                    try {
                        Number n = Expression.line(scanner);
                        System.out.println(n.floatValue());
                    } catch (java.lang.Exception e) {
                        System.err.println(scanner +": " + e);
                        while (scanner.ttype != scanner.TT_EOL
                           && scanner.nextToken() != scanner.TT_EOF)
                            ;
                    }
                while (scanner.ttype == scanner.TT_EOL);
            } catch (IOException ioe) { System.err.println(ioe); }
        }

        /** indicates parsing errors.
        */
        public static class Exception extends java.lang.Exception {
            public Exception (String msg) {
                super(msg);
            }
        }

        /** recognizes line: sum '\n';
         an empty line is silently ignored.
         @param s source of first input symbol, may be at end of file.
         @return tree for sum, null if only end of file is found.
         @throws Exception for syntax error.
         @throws IOException discovered on s.
         */
        public static Number line (Scanner s) throws Exception, IOException {
            for (; ; )
        }
```
switch (s.nextToken()) {
    default:
        Number result = sum(s);
        if (s.ttype != eol) throw new Exception("expecting nl");
        return result;
    case eol: continue;  // ignore empty line
    case eof: return null;
}

/** recognizes product: term [{ ('*'|'%'|'/') term }];
 * @param s ource of first input symbol, advanced beyond product.
 * @return tree with evaluators.
 * @see Expression#sum
 */
public static Number product (Scanner s) throws Exception, IOException {
    Number result = term(s);
    for (;;)
        switch (s.ttype) {
            case '*':
                s.nextToken();
                result = new Node.Mul(result, term(s));
                continue;
            case '/':
                s.nextToken();
                result = new Node.Div(result, term(s));
                continue;
            case '%':
                s.nextToken();
                result = new Node.Mod(result, term(s));
                continue;
            default:
                return result;
        }

    /** recognizes sum: product [{ ('+'|'-'|') product }];
     * @param s ource of first input symbol, advanced beyond sum.
     * @return tree with evaluators.
     * @see Expression#line
     */
    public static Number sum (Scanner s) throws Exception, IOException {
        Number result = product(s);
        for (;;)
            switch (s.ttype) {
                case '+':
                    s.nextToken();
                    result = new Node.Add(result, product(s));
                    continue;
                case '-':
                    s.nextToken();
                    result = new Node.Sub(result, product(s));
                    continue;
                default:
                    return result;
            }
/** recognizes term: '('sum')' | Number;
   @param s source of first input symbol, advanced beyond term.
   @return tree with evaluators.
   @see Expression#sum */

public static Number term (Scanner s) throws Exception, IOException {
   switch (s.ttype) {
      case '(':
         s.nextToken();
         Number result = sum(s);
         if (s.ttype != ')') throw new Exception("expecting ")
         s.nextToken();
         return result;
      case word:
         result = s.sval.indexOf( ")" < 0 ? (Number)new Long(s.sval)
         s.nextToken(); return result;
      throw new Exception("missing term");
   }
} // end of class Expression

Source Code: Src/9_e/Expression.java

/*
 * Thanks to ats
 */

import java.io.Serializable;

/** base class to store and evaluate arithmetic expressions.
   Defines most value-functions so that subclasses need only deal
   with long and double arithmetic.
 */

public abstract class Node extends Number implements Serializable {

   /** maps byte arithmetic to long.
      @return truncated long value.
   */
   public byte byteValue () {
      return (byte)longValue();
   }

   /** maps short arithmetic to long.
      @return truncated long value.
   */
   public short shortValue () {
      return (short)longValue();
   }

   /** maps int arithmetic to long.
      @return truncated long value.
   */
   public int intValue () {
      return (int)longValue();
   }

   /** maps long arithmetic to long.
      @return truncated long value.
   */
   public long longValue () {
      return (long)longValue();
   }

   /** maps double arithmetic to long.
      @return truncated long value.
   */
   public double doubleValue () {
      return (double)longValue();
   }

   /** maps float arithmetic to long.
      @return truncated long value.
   */
   public float floatValue () {
      return (float)longValue();
   }

   /** maps boolean arithmetic to long.
      @return truncated long value.
   */
   public boolean booleanValue () {
      return (boolean)longValue();
   }

   // More functions...
public int intValue () {
    return (int)longValue();
}

/** maps float arithmetic to double.
   @return truncated double value. */
public float floatValue () {
    return (float)doubleValue();
}

/** represents a binary operator.
   Must be subclassed to provide evaluation. */
protected abstract static class Binary extends Node {
    /** left operand subtree.
     *@serial left operand subtree. */
    protected Number left;

    /** right operand subtree.
     *@serial right operand subtree. */
    protected Number right;

    /** builds a node with two subtrees.
     *@param left left subtree.
     *@param right right subtree. */
    protected Binary (Number left, Number right) {
        this.left = left; this.right = right;
    }
}

/** represents a unary operator.
   Must be subclassed to provide evaluation. */
protected abstract static class Unary extends Node {
    /** operand subtree.
    *@serial operand subtree. */
    protected Number tree;

    /** builds a node with a subtree.
     *@param tree subtree. */
    protected Unary (Number tree) {
        this.tree = tree;
    }
}

/** implements addition. */
public static class Add extends Binary {
/** builds a node with two subtrees.
 * @param left left subtree.
 * @param right right subtree.
 */
public Add (Number left, Number right) {
    super(left, right);
}

/** implements long addition.
 * @return sum of subtree values.
 */
public long longValue () {
    return left.longValue() + right.longValue();
}

/** implements double addition.
 * @return sum of subtree values.
 */
public double doubleValue () {
    return left.doubleValue() + right.doubleValue();
}

/** implements subtraction.
 */
public static class Sub extends Binary {
    /** builds a node with two subtrees.
     * @param left left subtree.
     * @param right right subtree.
     */
    public Sub (Number left, Number right) {
        super(left, right);
    }

    /** implements long subtraction.
     * @return difference of subtree values.
     */
    public long longValue () {
        return left.longValue() - right.longValue();
    }

    /** implements double subtraction.
     * @return difference of subtree values.
     */
    public double doubleValue () {
        return left.doubleValue() - right.doubleValue();
    }
}

/** implements multiplication.
 */
public static class Mul extends Binary {
    /** builds a node with two subtrees.
     * @param left left subtree.
     */
    public Mul (Number left) {
        super(left);
    }

    /** implements long multiplication.
     * @return product of subtree values.
     */
    public long longValue () {
        return left.longValue() * right.longValue();
    }

    /** implements double multiplication.
     * @return product of subtree values.
     */
    public double doubleValue () {
        return left.doubleValue() * right.doubleValue();
    }
}
public Mul (Number left, Number right) {
    super(left, right);
}

/** implements long multiplication. 
* @return product of subtree values. 
*/
public long longValue () {
    return left.longValue() * right.longValue();
}

/** implements double multiplication. 
* @return product of subtree values. 
*/
public double doubleValue () {
    return left.doubleValue() * right.doubleValue();
}

/** implements division. 
*/
public static class Div extends Binary {
    /** builds a node with two subtrees. 
    @param left left subtree. 
    @param right right subtree. 
    */
    public Div (Number left, Number right) {
        super(left, right);
    }

    /** implements long division. 
    @return quotient of subtree values. 
    */
    public long longValue () {
        return left.longValue() / right.longValue();
    }

    /** implements double division. 
    @return quotient of subtree values. 
    */
    public double doubleValue () {
        return left.doubleValue() / right.doubleValue();
    }
}

/** implements modulus. 
*/
public static class Mod extends Binary {
    /** builds a node with two subtrees. 
    @param left left subtree. 
    @param right right subtree. 
    */
    public Mod (Number left, Number right) {
        super(left, right);
    }

    /** implements long modulus. 
    */
    public long longValue () {
        return left.longValue() % right.longValue();
    }

    /** implements double modulus. 
    */
    public double doubleValue () {
        return left.doubleValue() % right.doubleValue();
    }
}
public Mod (Number left, Number right) {
    super(left, right);
}

/** implements long modulus.
 * @return remainder after division of subtree values.
 */
public long longValue () {
    return left.longValue() % right.longValue();
}

/** implements double modulus.
 * @return remainder after division of subtree values.
 */
public double doubleValue () {
    return left.doubleValue() % right.doubleValue();
}

/** implements sign change.
 */
public static class Minus extends Unary {
    /** builds a node with a subtree.
     * @param tree subtree.
     */
    public Minus (Number tree) {
        super(tree);
    }

    /** implements long sign change.
     * @return negative of subtree value.
     */
    public long longValue () {
        return -tree.longValue();
    }

    /** implements double sign change.
     * @return negative of subtree values.
     */
    public double doubleValue () {
        return -tree.doubleValue();
    }
}

Source Code: Src/9_e/Node.java

Result:
% java Expression
2 * ( 1 - 2 )
-2.0
11.22. Not Discussed.

- **PipedInputStream**
  A piped input stream should be connected to a piped output stream; the piped input stream then provides whatever data bytes are written to the piped output stream. Typically, data is read from a PipedInputStream object by one thread and data is written to the corresponding PipedOutputStream by some other thread. Attempting to use both objects from a single thread is not recommended, as it may deadlock the thread. The piped input stream contains a buffer, decoupling read operations from write operations, within limits.

- **PushbackInputStream**
  A PushbackInputStream adds functionality to another input stream, namely the ability to "push back" or "unread" one byte. This is useful in situations where it is convenient for a fragment of code to read an indefinite number of data bytes that are delimited by a particular byte value; after reading the terminating byte, the code fragment can "unread" it, so that the next read operation on the input stream will reread the byte that was pushed back. For example, bytes representing the characters constituting an identifier might be terminated by a byte representing an operator character; a method whose job is to read just an identifier can read until it sees the operator and then push the operator back to be re-read.

- **DigestInputStream**
  A transparent stream that updates the associated message digest using the bits going through the stream. To complete the message digest computation, call one of the digest methods on the associated message digest after your calls to one of this digest input stream’s read methods.

- **JCE (Java Crypto Package)**

- **ZipIn/OutputStream**
  This class implements an input stream filter for reading files in the ZIP file format. Includes support for both compressed and uncompressed entries.

  - and more .....  

12. Threads

See also: and

12.1. Intro

Java programs, applications, and applets can consists of threads which conceptually are executed in parallel. This section demonstrates with simple examples how threads are created and manipulated, how exclusive access to common variables is managed, how conditional access is obtained, and how threads are connected with pipelines. Two classical examples concern communication with semaphores and conditional access to resources.
12.2. Principles and Features

- A thread is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.

- Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority.

- Each thread may or may not also be marked as a daemon. The Java Virtual Machine exits when the only threads running are daemon threads.

- When code running in some thread creates a new Thread object, the new thread has its priority initially set equal to the priority of the creating thread, and is a daemon thread if and only if the creating thread is a daemon, unless specified otherwise.

- As is everything else, during construction the object is controlled by method; afterwards its execution can be controlled through `start()`, `setPriority()`, `...` investigated with `getPriority()` and `isAlive()`. JDK 1.1 is supposed to implement interruptions as well.
12.3. Creation and Using

• Threads can be created using an instance of a class which is a subclass of Thread.
• Threads can be created using an instance of a class that implements the interface.

Why are there two different ways?

```java
public interface Runnable {
    public abstract void run();
}
```

Example: run

```java
1 2 public class Thread_1 extends Thread {
3     private String info;
4     static int x = 0;
5
6     public Thread_1 (String info) {
7         this.info = info;
8     }
9
10    public void run () {
11       if ( info == 1 )
12       x = 3;
13       else
14         x = 1;
15       System.out.print(x);
16    }
17
18    public static void main (String args []) {
19       Thread_1 aT1 = new Thread_1(1);
20       Thread_1 aT2 = new Thread_1(2);
21       aT1.start();
22       aT2.start();
23       System.out.println(x);
24    }
25
26 }
```

Source Code: Src/11/Thread_1.java

Result:

```
% java Thread_1 a b c d e f g h i j k l m n o p q r
bdfhjnpracegikmoqs%
```

Example: runnable

```java
1 2 public class Thread_1b implements Runnable {
3
4     private String info;
5     int x = 0;
6
```
```java
public Thread_1b (String info) {
    this.info = info;
}

public void run () {
    x=1;
    System.out.print(info);
}

public static void main (String args []) {
    if (args != null) {
        for (int n = 0; n < args.length; ++ n) {
            new Thread( new Thread_1b("" + n ) ).start();
        }
    }
}

Source Code: Src/11/Thread_1b.java
```
public class Thread_2 extends Thread {
    private String info;

    public Thread_2 (String info) {
        this.info = info;
    }

    public void run () {
        long sleep = (int)(Math.random() * 10000);
        System.out.println(info + " sleeps for " + sleep );
        try {
            sleep(sleep);
        } catch (InterruptedException e) {
            e.getMessage();
        }
    }

    public static void main (String args []) {
        int count = 0;
        if (args != null)
            for (int n = 0; n < args.length; ++ n) {
                Thread_2 aT1 = new Thread_2(args[n]);
                if ( n % 2 == 0 )
                    aT1.setPriority(Thread.MIN_PRIORITY);
                aT1.start();
            }
        while (count != 1 ) {
            try {
                count = activeCount();
                System.out.println("activeCount() = " + count );
                sleep(500); 
            } catch (InterruptedException e) {
                e.getMessage();
            } 
        }
    }
}
java Thread_2 a b c d
activeCount() = 5
b sleeps for 1063
d sleeps for 8295
a sleeps for 2197
c sleeps for 2619
activeCount() = 5
activeCount() = 5
activeCount() = 4
activeCount() = 4
activeCount() = 3
activeCount() = 2
...
activeCount() = 2
activeCount() = 1
Stolen from Java doc (sun)

- Threads belong to groups, represented as ThreadGroup objects and ordered hierarchically. Through a group, several threads can be controlled together; e.g., a group defines a maximum priority for its members. As a thread can only influence threads in its own group, the extent of a thread’s influence on others can be limited.

- Threads and groups have names, which, however, are mostly useful for documentation and debugging. Theoretically, the classes can be used to identify all threads and all groups; this can be used to build a thread variant of `ps(1)`.

- Internally, the Java runtime system uses several threads that deal, for example, with unreachable objects (garbage collection). If a main program is started, this takes place in the thread main. Once a window is opened, more threads are added.

The execution of the Java Virtual Machine ends once most threads reach the end of their run() methods. “Most” means that there are user and daemon threads and thread groups; execution of the JVM ends when there are no more user threads. The distinction is made by calling `setDaemon()`; the distinction is necessary, because threads that deliver events and manage painting in a window system are irrelevant for deciding if an application has completed its job.
12.4. Interruption

from:

```java
if (Thread.interrupted()) // Clears interrupted status!
    throw new InterruptedException();
```

```java
public class Thread_3 extends Thread {
    private String info;
    Thread_3 aT1;

    public Thread_3 (String info) {
        this.info = info;
    }

    public void run () {
        System.out.println(info + " is running");
        try {
            sleep(1000000);
        } catch (InterruptedException e) {
            System.err.println("Interrupted!");
            if (isInterrupted())
                System.err.println("yup it’s true.");
        }
        System.out.println(info + ": exit run");
    }

    public static void main (String args []) {
        Thread_3 aT1 = new Thread_3("first");
        aT1.start();
        System.err.println("interrupt ‘first’");
        aT1.interrupt();
    }
}
```

Source Code: Src/11/Thread_3.java
Result:

% java Thread_3
first is running
interrupt 'first'
Interrupted!
first: exit run

Extract from Javadoc:

The interrupted status of the current thread is cleared when this exception is thrown.
12.5. Join

```java
public class Join extends Thread {
    private String info;
    Join aT1;

    public Join (String info) {
        this.info = info;
    }

    public void run () {
        System.out.println(info + " is running");
        try {
            sleep(10000);
        }
        catch (InterruptedException e) {
            System.err.println("Interrupted!");
            try {
                sleep(100);
            } catch (Exception ee) {} 
        }
        System.out.println(info + ": exit run");
    }

    public static void main (String args []) {
        Join aT1 = new Join("first");
        aT1.start();
        try { sleep(100); } catch (Exception e) { e.printStackTrace(); }
        aT1.interrupt();
        if (Thread.interrupted())
            System.err.println("A thread got interrupted");
        try {
            aT1.join();
            System.err.println("Got it");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.err.println("main end");
    }
}
```

Source Code: Src/11/Join.java
Result:

first is running
third is running
second is running
second: exit run
third: exit run
first: exit run
Got it

Extract from Javadoc:

The interrupted status of the current thread is cleared when this exception is thrown.
12.6. Join II

```java
public class Evaluator extends Thread {
    public void run() {
        System.out.println("---->");
        // try { sleep(1000000); } catch (InterruptedException e) { }
        System.out.println("<----");
    }

    public void multiply() {
        for (int i = 0; i < 10; i++) {
            for (int j = 0; j < 10; j++) {
                // Evaluator et = new Evaluator();
                Thread et = new Evaluator();
                et.start();
                /*
                try {
                    et.join();
                }*/
                catch (InterruptedException e) {
                    System.out.println("Interrupted!");
                }
            }
        }
    }

    public static void main(String[] args) {
        Evaluator eval = new Evaluator();
        eval.multiply();
    }
}
```

Source Code: Src/11/Evaluator.java
Is the following output possible:

```java
% java Evaluator
----->
----->
----->
<---->
<---->
<---->
----->
----->
...
```

12.7. Using the Interface Runnable

```java
import java.util.*;

public class Thread_R implements Runnable {

    private String name;
    private Vector aVector;

    public Thread_R (String name) {
        this.name = name;
    }

    public void run () {
        System.out.println("Hi :) ... my name is: " + name);
    }

    public static void main (String args []) {
        String names[] = { "bono", "U2" };
        for ( int index = 0; index < names.length; index ++ ) {
            new Thread( new Thread_R( names[index] ) ).start();
        }
    }
}
```

Source Code: Src/11/Thread_R.java

12.8. Thread States
Thread created

new

start

ready

run

yield

completed

running

inactive

sleep, or wait

notify, notifyAll
import java.util.*;

public class CoresTest extends Thread {
    static final int soManyThreads = Runtime.getRuntime().availableProcessors();
    static final int soOftenPerThread = 1000000;
    static long milliSeconds = 0;
    double result = 0;
    int id;

    public CoresTest(int index) {
        id = index;
    }

    public static void init() {
        milliSeconds = System.currentTimeMillis();
    }

    public static void end(String s) {
        System.err.println(s + " : " + (System.currentTimeMillis() - milliSeconds) + 
                           " # of cores : " + 
                           Runtime.getRuntime().availableProcessors()
        );
    }

    public static void main (String args []) {
        CoresTest single = new CoresTest(0);
        CoresTest[] many = new CoresTest[soManyThreads];
        CoresTest o = null;
        init();
        single.singleThreadTest(soOftenPerThread);
        end("Single Thread Test");

        init();
        for ( int index = 0; index < soManyThreads; index ++ ) {
            many[index] = new CoresTest(index);
            many[index].start();
        }
        try {
            for ( int index = 0; index < soManyThreads; index ++ ) {
                many[index].join();
            }
        } catch (Exception e ) {
            e.printStackTrace();
        }
        end("Multiple Core Test");
    }

}
12.10. Competing Threads

Threads can obtain exclusive access to an object if all competing threads use a synchronized statement or call a method with synchronized attribute. Class methods monitor the class description, other methods monitor their receiver, the statement monitors the indicated value. The attribute synchronized precedes the result type.

Adding or deleting a synchronized modifier of a method does not break compatibility with existing binaries.

- The execution of methods can be synchronized.
  ```java
  public synchronized method( ...) { ... }
  ```

- Synchronized statements which allow access to an associated object.
  ```java
  synchronized(aObj) { ... }
  ```

  ```java
  public synchronized method( ...) { ... }
  ```

  Is equivalent to
  ```java
  public method( ...) {
      synchronized ( this ) {
      }
  }
  ```

Note: Interface methods can’t be native, static, synchronized, final, private, or protected
12.11. Example I

```java
import java.util.*;

public class M extends Thread {
    private String info;
    private Vector aVector;

    public M (String info) {
        this.info = info;
    }

    private synchronized void inProtected () {
        System.err.println(info + " : is in protected()");
        try {
            sleep(1000);
        }
        catch (InterruptedException e ) {
            System.err.println("Interrupted!");
        }
        System.err.println(info + " : exit run");
    }

    public void run () {
        inProtected();
    }

    public static void main (String args []) {
        Vector aVector = new Vector();
        M aT4_0 = new M("first");
        aT4_0.start();
        aT4_0.inProtected();
    }
}
```

Source Code: Src/11/M.java

first: is in protected()
first: exit run
first: is in protected()
first: exit run

12.12. Example II

This example has a problem.

zwei unterschiedlich empfänger fuer die methode
import java.util.*;

public class Thread_4 extends Thread {
    private String info;
    private Vector aVector;

    public Thread_4 (String info, Vector aVector) {
        this.info = info;
        this.aVector = aVector;
    }

    private synchronized void inProtected () {
        System.err.println(info + " : is in protected()");
        aVector.addElement(info);
        try {
            if ( info.equals("second") )
                sleep(1000);
            else
                sleep(3000);
        } catch (InterruptedException e) {
            System.err.println("Interrupted!");
        }
        System.err.println(info + " : exit run");
    }

    public void run () {
        inProtected();
    }

    public static void main (String args []) {
        Vector aVector = new Vector();
        Thread_4 aT4_0 = new Thread_4("first", aVector);
        Thread_4 aT4_1 = new Thread_4("second", aVector);
        aT4_0.start();
        aT4_1.start();
    }
}

Source Code: Src/11/Thread_4.java
Result:

first: is in protected()
second: is in protected()
second: exit run
first: exit run

the only possible output?
12.13. Object Synchronization

```java
import java.util.*;

public class Thread_5 extends Thread {
    private String info;
    private Vector aVector;

    public Thread_5 (String info, Vector aVector) {
        this.info = info;
        this.aVector = aVector;
    }

    public void inProtected () {
        synchronized (aVector) {
            System.err.println(info + " : ");
            try {
                if (info.equals("second") )
                    sleep(1000);
                else
                    sleep(3000);
            } catch (InterruptedException e) {
                System.err.println("Interrupted!");
            }
            System.err.println(info + ": exit run");
        }
    }

    public void run () {
        inProtected();
    }

    public static void main (String args []) {
        Vector aVector = new Vector();
        Thread_5 aT5_0 = new Thread_5("first", aVector);
        Thread_5 aT5_1 = new Thread_5("second", aVector);
        aT5_0.start();
        aT5_1.start();
    }

    Source Code: Src/11/Thread_5.java
```
Is

% java Thread_5
first: is in protected()
first: exit run
second: is in protected()
second: exit run

the only possible output?

12.14. Object Synchronization II

```java
import java.util.*;

public class Thread_5b extends Thread {
    private String info;
    private Vector aVector = new Vector();

    public Thread_5b (String info) {
        this.info = info;
    }

    public void inProtected () {
        synchronized (aVector) {
            System.err.println(info + " : is in protected()" );
            try {
                sleep(3000);
            } catch (InterruptedException e) {
                System.err.println("Interrupted!");
                System.err.println(info + " : exit run");
            }
        }
    }

    public void run () {
        inProtected();
    }

    public static void main (String args []) {
        Thread_5b aT5_0 = new Thread_5b("first");
        Thread_5b aT5_1 = new Thread_5b("second");
        aT5_0.start();
        aT5_1.start();
    }
}
```

Source Code: Src/11/Thread_5b.java
12.15. Object Synchronization III

What problem do you see?
How to fix it?

Das zu synchronisierende Object is zweimal vorhangen

```java
import java.util.*;

public class Thread_5c extends Thread {
    private String info;
    static Vector aVector;

    public Thread_5c (String info) {
        this.info = info;
    }

    public void inProtected () {
        synchronized ( aVector ) {
            System.err.println(info + " : in protected()");
            try {
                sleep(100);
            } catch (InterruptedException e) {
                System.err.println("Interrupted!");
            }
            System.err.println(info + ": exit run");
        }
    }

    public void run () {
        inProtected();
    }

    public static void main (String args []) {
        Vector aVector = new Vector();
        Thread_5c aT5_0 = new Thread_5c(aVector, "first");
        aT5_0.start();

        aVector = new Vector();
        Thread_5c aT5_1 = new Thread_5c(aVector, "second");
        aT5_1.start();
    }
}
```

Source Code: Src/11/Thread_5c.java
Is

% java Thread_5b
...

the only possible output?

12.16. Class Synchronization

```
import java.util.*;

public class ClassT extends Thread {
    private String info;
    private Vector aVector;

    public ClassT (String info, Vector aVector) {
        this.info = info;
        this.aVector = aVector;
    }

    static synchronized void staticInProtected1(String s) {
        System.err.println(s + ": ---->");
        try {
            sleep(1000);
        } catch (InterruptedException e) {
            System.err.println("Interrupted!");
        }
        staticInProtected2(s);
        System.err.println(s + ": <----");
    }

    static synchronized void staticInProtected2(String s) {
        System.err.println(s + ": ===>");
        try {
            sleep(1000);
        } catch (InterruptedException e) {
            System.err.println("Interrupted!");
        }
        System.err.println(s + ": ===>");
    }

    public void run () {
        staticInProtected1(info);
    }

    public static void main (String args []) {
        Vector aVector = new Vector();
        ClassT aClassT_0 = new ClassT("first", aVector);
        ClassT aClassT_1 = new ClassT("second", aVector);
        ClassT.staticInProtected1("main");
        aClassT_0.start();
    }
```
aClassT_1.start();
aClassT_0.staticInProtected1("aClassT_0");
aClassT_1.staticInProtected1("aClassT_1");
}

Source Code: Src/11/ClassT.java
% java ClassT
main: ---->
main: =====>
main: =====>
main: <----
aClassT_0: ---->
aClassT_0: =====>
aClassT_0: =====>
aClassT_0: <----
aClassT_1: ---->
aClassT_1: =====>
aClassT_1: =====>
aClassT_1: <----
first: ---->
first: =====>
first: =====>
first: <----
second: ---->
second: =====>
second: =====>
second: <----
import java.util.*;

public class Thread_6 extends Thread {
    private String info;
    private Vector aVector;

    public Thread_6 (String info, Vector aVector) {
        this.info = info;
        this.aVector = aVector;
    }

    private void inProtected_1 () {
        synchronized (aVector) {
            System.err.println("1: "+info+": is in ");
            try {
                sleep(1000);
            } catch (InterruptedException e) {
                System.err.println("Interrupted!");
            }
            System.err.println("1: "+info+": exit");
        }
    }

    private void inProtected_2 () {
        synchronized (info) {
            System.err.println("2: "+info+": is IN ");
            try {
                sleep(5000);
            } catch (InterruptedException e) {
                System.err.println("Interrupted!");
            }
            System.err.println("2: "+info+": EXIT");
        }
    }

    private static void inProtected_3 () {
        System.err.println("3: IN ");
        try {
            sleep(9000);
        } catch (InterruptedException e) {
            System.err.println("Interrupted!");
        }
        System.err.println("3: EXIT");
    }

    public void run () {
        inProtected_1();
        inProtected_2();
    }
}
public static void main (String args []) {
    Vector aVector = new Vector();
    Thread_6 aT6_0 = new Thread_6("first", aVector);
    Thread_6 aT6_1 = new Thread_6("second", aVector);
    aT6_0.start();
    aT6_1.start();
}
12.17. Wait and Notify

By using \texttt{wait} and \texttt{notify} a thread can give up its lock at an arbitrary point and the wait for another thread to give it back for continuation.

```java
import java.util.Vector;

public class WaitAndNotify_First extends Thread {
    private static int counter = 0;
    private String name = null;
    private Vector aVector;

    public WaitAndNotify_First (String name, Vector aVector) {
        this.aVector = aVector;
        this.name = name;
    }

    public synchronized void run () {
        if ( name.equals("one") ) {
            System.out.println(name + " will wait ...");
            aVector.notify();
            System.out.println(name + " done.");
        } else {
            System.out.println(name + " will wait ...");
            try {
                aVector.wait();
            } catch (IllegalMonitorStateException e) {
                System.out.println(" : IllegalMonitorStateException");
            } catch (InterruptedException e) {
                System.out.println(" : InterruptedException");
            }
            System.out.println(name + " is awake!");
        }
    }

    public static void main (String args []) {
        Vector theVector = new Vector();
        new WaitAndNotify_First("one", theVector).start();
        new WaitAndNotify_First("two", theVector).start();
    }
}
```

Source Code: Src/11/WaitAndNotify_First.java

How can we guarantee that the code behaves as expected ...

12.18. Wait and Notify II
import java.util.Vector;

public class WaitAndNotify_0 extends Thread {

    private static int counter = 0;
    static Vector aVector = new Vector();

    public WaitAndNotify_0 (String name, Vector aVector) {
        this.aVector = aVector;
        this.setName(name);
    }

    public void doTheJob() {
        synchronized (aVector) {
            counter ++;
            if (counter == 3) {
                System.out.println(getName() + " is waking up ...");
                aVector.notifyAll();
                System.out.println(getName() + " done.");
            } else {
                System.out.println(getName() + " will wait ...");
                try {
                    aVector.wait();
                } catch (IllegalMonitorStateException e) {
                    System.out.println(" : IllegalMonitorStateException");
                } catch (InterruptedException e) {
                    System.out.println(" : InterruptedException");
                }
                System.out.println(getName() + " is awake!");
            }
        }
    }

    public void run () {
        doTheJob();
    }

    public static void main (String args []) {
        new WaitAndNotify_0("eins", aVector).start();
        new WaitAndNotify_0("zwei", aVector).start();
        new WaitAndNotify_0("drei", aVector).start();
    }
}

Source Code: Src/11/WaitAndNotify_0.java

12.19. Wait and Notify II

What is wrong here: Ordnung - der letzte muss nicht der letzte sein.
import java.util.Vector;

public class WaitAndNotify extends Thread {

    private String info;
    static Vector aVector = new Vector();

    public WaitAndNotify (String info, Vector aVector) {
        this.info = info;
        this.aVector = aVector;
    }

    public void doTheJob() {
        synchronized (aVector) {
            if (info.equals("last")) {
                System.out.println(info + " is waking up ...");
                aVector.notifyAll();
                System.out.println(info + " done.");
            } else {
                System.out.println(info + " is waiting");
                try {
                    aVector.wait();
                } catch (IllegalMonitorStateException e) {
                    System.out.println(info + ": IllegalMonitorStateException");
                } catch (InterruptedException e) {
                    System.out.println(info + ": InterruptedException");
                }
                System.out.println(info + " is awake!");
            }
        }
    }

    public void run () {
        doTheJob();
    }

    public static void main (String args []) {
        new WaitAndNotify("first", aVector).start();
        new WaitAndNotify("second", aVector).start();
        new WaitAndNotify("last", aVector).start();
    }

}
Result:

% java WaitAndNotify
first is waiting
second is waiting
last is waking up ...
last done.
first is awake!
second is awake!
public class WaitAndNotify_2 extends Thread {

    private String info;
    static Integer monitor = new Integer(3);
    static int count = 0;
    static int max = 0;

    public WaitAndNotify_2 (String info) {
        this.info = info;
        // max ++;
    }

    public void doTheJob() {
        synchronized (monitor) {
            System.out.println(info + " is waiting");
            count ++;
            // if ( count == max );
            if ( count == 3 )
                monitor.notifyAll();
            else
                try {
                    monitor.wait();
                    sleep(1000);
                } catch ( Exception e ) {
                    System.out.println(info + ": IllegalMonitorStateException");
                }
            System.out.println(info + " is awake!");
        }
    }

    public void run () {
        doTheJob();
    }

    public static void main (String args []) {
        new WaitAndNotify_2("first").start();
        new WaitAndNotify_2("second").start();
        new WaitAndNotify_2("last").start();
    }
}

Source Code: Src/11/WaitAndNotify_2.java
Result:
% java WaitAndNotify_2
first is waiting
second is waiting
last is waiting
last is awake!
first is awake!
second is awake!

12.21. Be careful with wait(long timeout)

```java
1 import java.util.Vector;
2 import java.util.Date;
3
4 public class WaitAndNotify_3 extends Thread {
5     private String info;
6     static Vector aVector = new Vector();
7     public WaitAndNotify_3(String info, Vector aVector) {
8         this.info = info;
9         this.aVector = aVector;
10     }
11     public void doTheJob() {
12         synchronized (aVector) {
13             System.out.println(info + " is waiting. " + new Date());
14             try {
15                 aVector.wait(1000);
16             } catch (Exception e) {
17                 System.out.println(info + ": Exception");
18                 e.printStackTrace();
19             }
20             System.out.println(info + " is awake! " + new Date());
21         }
22     }
23
24     public void run() {
25         doTheJob();
26     }
27
28     public static void main(String args[]) {
29         new WaitAndNotify_3("first", aVector).start();
30         new WaitAndNotify_3("second", aVector).start();
31         // new WaitAndNotify_3("last", aVector).start();
32     }
33 }
```

Source Code: Src/11/WaitAndNotify_3.java
% java WaitAndNotify_3
first is waiting. Mon Apr 16 15:02:10 EDT 2001
second is waiting. Mon Apr 16 15:02:11 EDT 2001
first is awake! Mon Apr 16 15:02:12 EDT 2001
second is awake! Mon Apr 16 15:02:12 EDT 2001
%
12.22. Lock Objects

See also:

TBD

Stole from java doc:

Interfaces and classes providing a framework for locking and waiting for conditions that is distinct from built-in synchronization and monitors. The framework permits much greater flexibility in the use of locks and conditions, at the expense of more awkward syntax. The Lock interface supports locking disciplines that differ in semantics (reentrant, fair, etc), and that can be used in non-block-structured contexts including hand-over-hand and lock reordering algorithms. The main implementation is ReentrantLock.

The ReadWriteLock interface similarly defines locks that may be shared among readers but are exclusive to writers. Only a single implementation, ReentrantReadWriteLock, is provided, since it covers most standard usage contexts. But programmers may create their own implementations to cover nonstandard requirements.

The Condition interface describes condition variables that may be associated with Locks. These are similar in usage to the implicit monitors accessed using Object.wait, but offer extended capabilities. In particular, multiple Condition objects may be associated with a single Lock. To avoid compatibility issues, the names of Condition methods are different from the corresponding Object versions.

12.23. Executors

Copied from:

• An object that executes submitted Runnable tasks.
• This interface provides a way of decoupling task submission from the mechanics of how each task will be run, including details of thread use, scheduling, etc.
• An Executor is normally used instead of explicitly creating threads. For example, rather than invoking new Thread(new(RunnableTask())).start() for each of a set of tasks, you might use:

Stolen from java doc:

In all of the previous examples, there’s a close connection between the task being done by a new thread, as defined by its Runnable object, and the thread itself, as defined by a Thread object. This works well for small applications, but in large-scale applications, it makes sense to separate thread management and creation from the rest of the application. Objects that encapsulate these functions are known as executors. The following subsections describe executors in detail.

12.24. Executors: Interfaces

TBD

12.25. Executors: Thread Pools

• Using thread pools avoids the overhead of creating threads
• This reduces also memory management overheads


• Help to make usage of multiple processors
• Designed for problems which have recursive solutions

12.27. Examples
public class T_1 extends Thread {
    private synchronized void inProtected () {
        System.err.println("--> ");
        try {
            sleep(1000);
        } catch (InterruptedException e) {
            System.err.println("Interrupted!");
        }
        System.err.println("<-- ");
    }
    public void run () {
        inProtected();
    }
    public static void main (String args []) {
        new T_1().start();
        new T_1().start();
        new T_1().start();
    }
}

Source Code: Src/Question_Week_5/T_1.java

public class T_2 extends Thread {
    private static String info;
    public T_2(String info ) {
        this.info = info;
    }
    private void inProtected () {
        synchronized (info) {
            System.err.println("--> " + info);
            try {
                sleep(1000);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        System.err.println("<-- " + info);
    }
}
```java
public void run () {
    inProtected();
}

public static void main (String args []) {
    T_2 one = new T_2("a");
    one.start();
    T_2 two = new T_2("b");
    two.start();
    // new T_2("a").start();
    // new T_2("b").start();
}

Source Code: Src/Question_Week_5/T_2.java

public class T_3 extends Thread {
    private int info;

    public T_3 (int info) {
        this.info = info;
    }

    public synchronized void run () {
        System.err.println("--> " + info);
        try {
            sleep(1000);
        } catch ( Exception e ) {
            e.printStackTrace();
        }
        System.err.println("<-- " + info);
    }

    public static void main (String args []) {
        for ( int i = 1; i < 100; i ++ )
            new T_3(i).start();
    }

Source Code: Src/Question_Week_5/T_3.java
```
public class T_4 extends Thread {

    static Object o = new Object();
    String info;

    public T_4(String info) {
        this.info = info;
    }

    public void run() {
        synchronized (o) {
            System.err.println("-->" + info);
            try {
                sleep(1000);
            } catch (InterruptedException e) {
                System.err.println("Interrupted!");
            }
            System.err.println("<---" + info);
        }
    }

    public static void main(String args[]) {
        new T_4("1").start();
        new T_4("2").start();
        new T_4("3").start();
    }
}

Source Code: Src/Question_Week_5/T_4.java

public class T_5 extends Thread {
    static String i = "2";
    static String theValue;

    T_5(String i) {
        this.i = i;
    }

    public void run() {
        if (this.i.equals("1"))
            theValue = "3";
        else
            theValue = "4";
    }
}

Source Code: Src/Question_Week_5/T_5.java
public static void main (String args []) {
    T_5 aT_5_a = new T_5("1");
    T_5 aT_5_b = new T_5("2");
    aT_5_a.start();
    aT_5_a.run();
    System.out.println("aT_5_a.i = " + aT_5_a.i );
    System.out.println("aT_5_b.i = " + aT_5_b.i );
}

Source Code: Src/Question_Week_5/T_5.java

public class T_6 extends Thread {
    String i = "2";
    static String theValue;
    T_6(String i) {
        this.i = i;
    }
    public void run () {
        if (this.i.equals("1") )
            theValue = "3";
        else
            theValue = "4";
    }
    /*
     * falls run nichts passiert in parallell
     */
    public static void main (String args []) {
        T_6 aT_6_1 = new T_6("1");
        T_6 aT_6_2 = new T_6("1");
        aT_6_1.start();
        aT_6_2.start();
        System.out.println("the value = " + theValue );
    }
}

Source Code: Src/Question_Week_5/T_6.java

public class T_7 extends Thread {
    String i = "2";
    static String theValue;
    T_7(String i) {
        this.i = i;
    }
    public void run () {
        synchronized (theValue ) {
            if (this.i.equals("1") )
                theValue = "3";
            else
                theValue = "4";
        }
    }
}

Source Code: Src/Question_Week_5/T_7.java
public class T_7 extends Thread {
    static String i = null;
    T_7(String i) {
        this.i = i;
    }
    public void run () {
        synchronized ( i ) {
            if ( this.i.equals("1") )
                i = "3";
            else
                i = "4";
        }
    }
}

public static void main (String args []) {
    T_7 aT_7_1 = new T_7("1");
    T_7 aT_7_2 = new T_7("1");
aT_7_1.start();
aT_7_2.start();
    synchronized ( theValue ) {
        System.out.println("theValue = " + theValue );
    }
}

Source Code: Src/Question_Week_5/T_7.java

public class T_8 extends Thread {
    static String i = null;
    T_8(String i) {
        this.i = i;
    }
    public void run () {
        synchronized ( i ) {
            if ( this.i.equals("1") )
                i = "3";
            else
                i = "4";
        }
    }
}

public static void main (String args []) {
    T_8 aT_8 = new T_8("1");
aT_8.start();
    System.out.println("aT_8.i = " + aT_8.i );
    System.out.println("aT_8.i = " + aT_8.i );
}

Source Code: Src/Question_Week_5/T_8.java

/*
 * is this output 1 0 1 0 1 ...
 * the only possible output?
 * Falsch: es ist nichtgarantiert, in welcher die 
 * Threads eintreten.
 */
public class T extends Thread {
    /*
     * is this output 1 0 1 0 1 ...
     * the only possible output?
     * Falsch: es ist nichtgarantiert, in welcher die
     * Threads eintreten.
     * /
     */
}

Source Code: Src/Question_Week_5/T.java
private String info;
static Object o = new Object();
public T (String info) {
    this.info = info;
}
public void run () {
synchronized (o) {
    while (true) {
        System.out.println(info);
        try {
            o.notify();
            sleep(100);
            o.wait();
        } catch (Exception e) {}
    }
}
public static void main (String args []) {
    (new T("0")).start();
    (new T("1")).start();
}

H extends Thread {
    String info = "";
    public H (String info) {
        this.info = info;
    }
    public synchronized void run () {
        try {
            while (true) {
                System.out.println(info);
                this.notify();
                this.wait();
            }
        } catch (Exception e) {} 
    }
    public static void main (String args []) {
        new H("0").start();
        new H("1").start();
    }
}

/*
* is this output 1 0 1 0 1 ...
* ...
* the only possible output?
*/
public class X extends Thread {
    private String info;
    static Object o = new Object();
    public X(String info) {
        this.info = info;
    }
    public void run() {
        while (true) {
            synchronized (o) {
                System.out.println(info);
                try {
                    o.notify();
                    sleep(100);
                    o.wait();
                } catch (Exception e) {
                }
            }
        }
    }
    public static void main(String args[]) {
        (new X("0")).start();
        (new X("1")).start();
    }
}

Source Code: Src/Question_Week_5/X.java

    public class ZeroOneOrSo extends Thread {
        private String info;
        static Object o = new Object();
        public ZeroOneOrSo(String info) {
            this.info = info;
        }
        public void run() {
            while (true) {
                synchronized (o) {
                    o.notify();
                    System.out.println(info);
                    try {
                        sleep(300);
                        o.wait();
                    } catch (Exception e) {
                    }
                }
            }
        }
    }

    public static void main(String args[]) {
        (new ZeroOneOrSo("0")).start();
        (new ZeroOneOrSo("1")).start();
    }
}
public static void main (String args []) {  
    new ZeroOneOrSo("0").start();  
    new ZeroOneOrSo("1").start();  
}  

Source Code: Src/Question_Week_5/ZeroOneOrSo.java

public class XXX extends Thread {  
    private String info;  
    static Object o = new Object();  
    
    public XXX (String info) {  
        this.info = info;  
        synchronized ( o ) {  
            if ( info.equals("0") )  
                ( new XXX("1") ).start();  
        }  
    }  
    
    public void run () {  
        while ( true ) {  
            synchronized ( o ) {  
                System.out.println(info);  
                try {  
                    o.notify();  
                    sleep(100);  
                    o.wait();  
                } catch ( Exception e ) { }  
            }  
        }  
    }  
    
    public static void main (String args []) {  
        new XXX("0").start();  
    }  
}  

Source Code: Src/Question_Week_5/XXX.java

12.28. Deadlock — an Overview

- Problem:

    Resource 1 and and resource 2 must be used exclusively
Process 1 holds resource 1 and is requesting resource 2

Process 2 holds resource 2 and is requesting resource 1

12.29. Deadlock

• A set of processes is in a deadlock state when every process in the set is waiting for an event that can be caused by only another process in the set.

12.30. Necessary Conditions

• A deadlock can occur if the following four conditions hold:
  — Mutual exclusion: At least one resource must be held in a non-sharable mode.
  — Hold and wait: There is a process that is holding a resource and is waiting to acquire another that is currently being held by other processes.
  — No preemption: Resources can only be released voluntarily.
  — Circular wait: See intro example.

12.31. Resource Graphs

12.32. Addressing Deadlock

• Prevention: Design the system so that deadlock is impossible.
• Avoidance: Construct a model of system states, then choose a strategy that will not allow the system to go to a deadlock state.
• Detection & Recovery: Check for deadlock (periodically or sporadically), then recover.

12.33. Prevention

• Necessary conditions for deadlock:
  — Mutual exclusion
  — Hold and wait
  — Circular waiting
  — No preemption

• Ensure that at least one of the necessary conditions is false at all times.
• Mutual exclusion must hold at all times (you can fudge things to get around this).

12.34. Hold and Wait

• Need to be sure a process does not hold one resource while requesting another.
• Approach 1: Force a process to request all resources it needs at one time (usually at startup). The process dies, if not all resources are available.
• Approach 2: If a process needs to acquire a new resource, it must first release all resources it holds, then reacquire all it needs.
• Problems:
  — Resource utilization may be low
  — Starvation is possible.
12.35. Circular Wait

Have a situation in which there are \( K \) processes holding units of \( K \) resources

- There is a cycle in the graph of processes and and resources
- Choose a resource request strategy by which no cycle will be introduced
- Total order on all resources, then can only ask for resources in numerical order (a minor variation is to merely insist that no process request a resource lower than what it is already holding).
- For example, if a set of resource types includes tape drives, disks, and printers, then the weights might be assigned as follows:
  - \( W(\text{tape drive}) = 1 \)
  - \( W(\text{disk drive}) = 5 \)
  - \( W(\text{printer}) = 12 \)
- If \( A \) needs tape, disk and printer and \( B \) needs printer and disk
  - \( A \rightarrow \text{tape}, \)
  - \( B \rightarrow \text{disk}, \)
  - \( B \rightarrow \text{printer}, \)
  - \( A \rightarrow \text{disk}, \)
  - \( A \rightarrow \text{printer} \)
- \( F \) should be defined in order of normal usage
- Proof by contradiction
  - Assume a circular wait exists
  - Let the set of processes involved in the circular wait be \( \{ P(0), P(1), P(2), \ldots, P(n) \} \), where \( P(i) \) is waiting for a resource \( R(i) \), which is held by process \( P((i+1)\% (n+1)) \) (modulo arithmetic is used on the indexes, so that \( P(n) \) is waiting on \( R(n) \) which is held by \( P(0) \))
  - Then, since \( P(i+1) \) is holding \( R(i) \) while requesting \( R(i+1) \), then \( W(R_i) < W(R_{i+1}) \) for all \( i \)
  - But this means that
    \[
    W(R_0) < W(R_1) < W(R_0)
    \]
- By transitivity, \( W(R_0) < W(R_0) \), which is impossible

12.36. Avoid Starvation and Deadlock

- Fairness is a problem, if several concurrent threads are competing for resources.
- A system is fair when each thread gets enough access to a limited resource to make a reasonable progress.
- A fair system prevents starvation and deadlock. Starvation occurs when one or more threads in your program is blocked from gaining access to a resource and thus cannot make progress.
- Deadlock is the ultimate form of starvation; it occurs when two or more threads are waiting on a condition that cannot be satisfied. Deadlock most often occurs when two (or more) threads are each waiting for the other(s) to do something.

12.37. DeadLocks

- Is there a deadlock in this program?
  und ist im synchronisierten block, bevor dem Aufruf inprtedcted_1 und fuer den ersten gilt das aequivalente, dann ist ein deadlock.
Is there a dead lock in this program?

Nein, aber ein StackOverflow wird eintreten.

•
private void inProtected_1 () {
    System.out.println(info + " tries to enter 1 .... ");
    synchronized (o) {
        System.out.println(info + " is in protected_1()");
        inProtected_2();
        System.out.println(info + ": exit inProtected_1");
    }
}

private void inProtected_2 () {
    System.out.println(info + " tries to enter 2 .... ");
    synchronized (o) {
        System.out.println(info + " is in protected_2()");
        inProtected_1();
        System.out.println(info + ": exit inProtected_2");
    }
}

public void run () {
    if (info.equals("first") )
        inProtected_1();
    else
        inProtected_2();
}

public static void main (String args []) {
    new DeadLock_2("second").start();
    try {
        sleep(1000);
    } catch (Exception e) {}
    new DeadLock_2("first").start();
}

Source Code: Src/11/DeadLock_2.java

Is there a dead lock in this program?

Angenommen jeder thread schafft es bis zum zweiten inProtected Aufruf, dann hat jeder die Ressource die der andere benoetigt.

12.38. Dining Philosophers

The dining philosophers are often used to illustrate various problems that can occur when many synchronized threads are competing for limited resources.

The story goes like this: Five philosophers are sitting at a round table. In front of each philosopher is a bowl of rice. Between each pair of philosophers is one chopstick. Before an individual philosopher can take a bite of rice he must have two chopsticks — one taken from the left, and one taken from the right. The philosophers must find some way to share chopsticks such that they all eat with an reasonable frequency.
import java.util.Random;
import java.util.Vector;

/** A class implementing the Dining Philosophers */
public class Philosopher extends Thread {

    protected static Random random = new Random(); // randomize
    protected int me; // number for trace
    protected Integer left, right; // my chopsticks

    public Philosopher (int me, Integer left, Integer right) {
        this.me = me; this.left = left; this.right = right;
    }

    /** philosopher’s body: think and eat 5 times */
    public void run () {
        for (int n = 1; n <= 5; ++ n) {
            System.out.println(me+" thinks");
            try {
                Thread.sleep((long)(random.nextFloat()*1000));
            } catch(Exception e) {
                e.printStackTrace();
            }
            System.out.println(me+" is trying to eat");
            synchronized (left) {
                synchronized (right) {
                    System.out.println("\t" + me+" eats");
                    try {
                        Thread.sleep((long)(random.nextFloat()*1000));
                    } catch(Exception e) {
                        e.printStackTrace();
                    }
                }
            }
            System.out.println("\t" + me+" leaves");
        }
    }

    /** sets up for 5 philosophers */
    public static void main (String args []) {
        Integer f[] = new Integer[5];
        for (int n = 0; n < 5; ++ n)
            f[n] = new Integer(n);
        Philosopher p[] = new Philosopher[5];
        p[0] = new Philosopher(0, f[4], f[0]); // backwards
        for (int n = 1; n < 5; ++ n)
            p[n] = new Philosopher(n, f[n-1], f[n]);
        for (int n = 0; n < 5; ++ n) p[n].start();
    }
}

Source Code: Src/11/Philosopher.java

What is wrong with this solution?
Angenommen es sind nur zwei Philosopher am Tisch un jeder greift nach seiner linken Gabel und is
erfolgreich, dann hat er keinen Zugriff zu rechten Gabel.
12.39. Semaphore

1965, suggested Edsger. W. Dijkstra to using an integer variable to count the number of wake ups: Semaphores (semaphore is a greek word, it stands for signal). A synchronization variable that take a positive integer variable. A semaphore has two operations:

- P (dutch for "to test", proberen): an atomic operation that waits for the semaphore to become positive, then decrements it by 1.
- V (dutch for "to increment", verhogen): an atomic operation that increments the semaphore by 1.

The P(S) operation on Semaphore S is:

If $S > 0$ then
    $S := S - 1$
else
    (Wait on S)

The V(S) operation on Semaphore S is:

If (One or more processes are waiting on S) then
    (Let one of the processes proceed)
else
    $S := S + 1$

It is assumed that P() and V() are indivisible.

If a thread tries to make a semaphore value to become negative, the thread is blocked until another thread makes the semaphore value positive.
public class Semaphore {
    protected int n;

    public Semaphore (int n) {
        this.n = n;
    }

    public synchronized void P () {
        if (n <= 0) {
            try {
                wait(); // see in object
            } catch(Exception e) {
                e.printStackTrace();
            }
        }
        -- n;
    }

    public synchronized void V () {
        if (++ n > 0)
            notify(); // see in object
    }
}

Source Code: Src/11/Semaphore.java

A thread blocks by calling wait(). Notification happens through notify() which, however, only releases one waiting thread from being blocked.
public class S extends Thread {
    protected int n;

    public S(int n) {
        this.n = n;
    }

    public synchronized void P() {
        if (n <= 0) {
            try {
                wait(); // see in object
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        -- n;
    }

    public synchronized void V() {
        if (++n > 0)
            notify(); // see in object
    }

    public void run() {
        for (int i = n; i >= 0; i -- ) {
            System.out.println("i = "+i);
            System.out.println("before P ");
            P();
            System.out.println("after P ");
        }
    }

    static public void main(String args[]) {
        S aS1 = new S(3);
        aS1.start();
        try {
            System.out.println("\tsleeping ...");
            sleep(1000);
            System.out.println("\tbefore V ");
            aS1.V();
            System.out.println("\tafter V ");
        }
        catch (InterruptedException e) {
            System.err.println("\tInterrupted!");
        }
    }
}

Source Code: Src/11/S.java
i = 3
before P
after P
i = 2
before P
after P
i = 1
before P
after P
i = 0
before P
sleeping ...
before V
after P
after V

12.41. Producer-Consumer Problem

Two processes share a common fixed size buffer. One of them, the producer, puts information into the buffer and the consumer takes it out.

Trouble arises when the producer wants to put a new item in the buffer, but it is already full. Similarly, if the consumer wants to remove an item from the buffer when the buffer is empty.

We will use three semaphores to solve this problem.

• full for counting the number of slots that are full.
• empty for counting the number of slots that are empty.
• mutex to make sure the producer and consumer do not access the buffer at the same time.

• full is initially 0
• empty is initially N
• mutex is initially 1 → no process is in its critical region.
public class Consumer extends Thread {
    final int N = 100;
    Semaphore mutex = new Semaphore(1); // access the buffer semaphore
    Semaphore empty = new Semaphore(N); // number of empty slots
    Semaphore full = new Semaphore(0); // number of used slots

    void insertItem(int i) {}
    void consumeItem(int i) {}
    void workWithIt(int i) {}

    void producer()
    {
        int item;
        while (true) {
            item = (int)Math.random();
            empty.P(); // decrement count of full slots
            mutex.P(); // enter critical region
            insertItem(item);
            mutex.V(); // leave critical region
            full.V(); // increment count of full slots
        }
    }

    void consumer()
    {
        int item = 3;
        while (true) {
            full.P(); // decrement count of full slots
            mutex.P(); // enter critical region
            consumeItem(item); // take it
            mutex.V(); // enter critical region
            empty.V(); // increment count of full slots
            workWithIt(item);
        }
    }
}

Source Code: Src/11/Consumer.java
public class Consumer_2 extends Thread {
    final int N = 100;
    Semaphoremutex = new Semaphore(1); // access the buffer semaphore
    Semaphore empty = new Semaphore(N); // number of empty slots
    Semaphore full = new Semaphore(0); // number of used slots

    void insertItem(int i) {}
    void consumeItem(int i) {}
    void workWithIt(int i) {}

    void producer() {
        int item;
        while (true) {
            item = (int)Math.random(); // enter critical region
            mutex.P(); // decrement count of full slots
            empty.P(); // decrement count of full slots
            insertItem(item);
        }
    }

    void consumer() {
        int item = 3;
        while (true) {
            full.P(); // enter critical region
            mutex.P(); // take it
            consumeItem(item);
            mutex.V(); // enter critical region
            empty.V(); // increment count of full slots
            workWithIt(item);
        }
    }
}

Source Code: Src/11/Consumer_2.java
SORRY, it’s only for me.

— Die beiden down-Operationen des Erzeugers sind ausgeführt.
— Falls der Puffer voll ist, wird der Erzeuger blockiert und mutex erhält den Wert 0.
— Wenn der Verbraucher das nächste Mal auf den Puffer zugreifen möchte, ist ihm dies nicht möglich → deadlock
12.42. Questions
What is going on here?

```java
public class T_1 extends Thread {

    private static synchronized void inProtected () {
        System.err.println("--> ");
        try {
            sleep(1000);
        } catch (InterruptedException e) {
            System.err.println("Interrupted!");
        }
        System.err.println(" <-- ");
    }

    public void run () {
        inProtected();
    }

    public static void main (String args []) {
        new T_1().start();
        new T_1().start();
        new T_1().start();
    }

    Source Code: Src/11q/T_1.java
```
public class T_2 extends Thread {
    private String info;

    public T_2(String info) {
        this.info = new String(info);
    }

    private void inProtected() {
        synchronized (info) {
            System.err.println("--> " + info);
            try {
                sleep(1000);
            } catch (Exception e) {
                e.printStackTrace();
            }
            System.err.println("<-- " + info);
        }
    }

    public void run() {
        inProtected();
    }

    public static void main(String args[]) {
        String a = "hello";
        new T_2(a).start();
        new T_2(a).start();
    }
}

Source Code: Src/11q/T_2.java

--> hello
--> hello
<-- hello
<-- hello
import java.util.Vector;

public class T_3 extends Thread {
    static Vector aVector = new Vector();
    private int info;

    public T_3(int info) {
        this.info = info;
    }

    public synchronized void run() {  
        System.err.println("--> "+info);
        try {
            sleep(1000);
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.err.println("<-- "+info);
    }  

    public static void main(String args[]) {
        for (int i = 1; i < 100; i++)
            new T_3(i).start();
    }
}

Source Code: Src/11q/T_3.java

--- 1
--- 2
--- 3
--- 4
--- 5
...
<-- 94
<-- 95
<-- 96
<-- 97
<-- 98
<-- 99

import java.util.*;

public class T_4_1 extends Thread {
    static Object o = new Object();

    public T_4_1() {
        o = new Object();
    }

    public void run() {
        synchronized (o) {

```java
try {
    sleep(1000);
} catch (InterruptedException e) {
    System.err.println("Interrupted!");
}
System.err.println("<---");

public static void main (String args []) {
    new T_4_1().start();
    new T_4_1().start();
    new T_4_1().start();
}
```

Source Code: Src/1q/T_4_1.java
```java
import java.util.*;

public class T_4 extends Thread {
    static Object o = new Object();

    public void run () {
        synchronized (o) {
            System.err.println("--->");
            try {
                sleep(1000);
            } catch (InterruptedException e) {
                System.err.println("Interrupted!");
            }
            System.err.println("<---");
        }

    public static void main (String args []) {
        new T_4().start();
        new T_4().start();
        new T_4().start();
    }

Source Code: Src/11q/T_4.java
```
import java.util.*;

public class T_5 extends Thread {
    static Object o = new Object();
    static int counter = 0;

    public void run () {
        if (++counter == 1 )
            o = new Object();

        synchronized (o) {
            System.err.println("---> ");
            try {
                sleep(1000);
            } catch (InterruptedException e) {
                System.err.println("Interrupted!");
            }
            System.err.println("<---");
        }
    }

    public static void main (String args []) {
        new T_5().start();
        new T_5().start();
        new T_5().start();
    }
}

Source Code: Src/1lq/T_5.java
```
import java.util.*;

public class T_6 extends Thread {
    static Object o = new Object();
    static int counter = 0;

    public void run () {
        if (counter++ == 1)
            o = new Object();

        synchronized (o) {
            System.err.println("--->");
            try {
                sleep(1000);
            } catch (InterruptedException e) {
                System.err.println("Interrupted!");
            }
            System.err.println("<---");
        }

    }

    public static void main (String args []) {
        new T_6().start();
        new T_6().start();
        new T_6().start();
    }
}
```

--->
--->
<---
<---
<---

Will the following program terminate?

```
import java.util.Vector;

public class T_1 extends Thread {
    private String info;
    Vector aVector;
    Vector bVector;

    public T_1 (String info, Vector aVector) {
        this.info = info;
        this.aVector = aVector;
    }

    public void run() {
        synchronized (aVector) {
            if (info.equals("last")) {
```

aVector.notifyAll();
} else {
    System.out.println(info + " is waiting");
    try {
        aVector.wait();
    } catch (Exception e) {
        System.out.println(info + ": InterruptedException");
    }
    System.out.println(info + " is awake!");
}
}

public static void main (String args []) {
    Vector aVector = new Vector();
    Vector bVector = new Vector();
    new T_1("first", aVector).start();
    new T_1("second", bVector).start();
    new T_1("last", bVector).start();
}

12.43. Questions

12.44. Questions from Students

import java.util.*;
public class M extends Thread {
    private String info;
    private Vector aVector;

    public M (String info) {
        this.info = info;
    }

    private synchronized void inProtected () {
        System.err.println(info + " :is in protected()");
        try {
            sleep(1000);
        } catch (InterruptedException e) {
            System.err.println("Interrupted!");
        }
        System.err.println(info + " :exit run");
    }

    public void run () {
        inProtected();
    }

    public static void main (String args []) {
        Vector aVector = new Vector();
        M aT4_0 = new M("first");
        M aT5_0 = new M("second");
    }
}
Q2. If the object is synchronized in the main method, what is the significance of doing so? In the code below, in the main method, the object is synchronized after the run() method is executed. What is the significance of doing this?

Q3. In what scenario will a running thread go to a ready state?

Q4. In the slide numbered 12.8, it says Interface methods cannot be synchronized. We have not understood that part.

Had
/*
* Should print out 0 1 0 1 0 1 ...
* */
public class ZeroOne extends Thread {
    public static void main(String args []) {
        T_7 aT_7_1 = new T_7("1");
        T_7 aT_7_2 = new T_7("1");
        aT_7_1.run();
        aT_7_2.run();
        synchronized (theValue ) {
            System.out.println("aT_7_1.theValue = " + aT_7_1.theValue);
            System.out.println("aT_7_2.theValue = " + aT_7_2.theValue);
        }
    }
}
private String info;
static Object o = new Object();
static boolean oneIsRunning = false; // is static important?

// es wird nur ein
// Objekt erzeugt

public ZeroOne (String info) {
    this.info = info;
}

public void run () {
    while (true) {
        synchronized (o) {
            o.notify();
            System.out.println(info);
            try {
                if (!oneIsRunning) {
                    (new ZeroOne("1")).start();
                    oneIsRunning = true;
                }
                sleep(300);
                o.wait();
                } catch (Exception e) { }
        }
    }
}

public static void main (String args []) {
    new ZeroOne("0").start();
}

Source Code: Src/Question_Week_5/ZeroOne.java
13. Networking

13.1. The Subject

Computer networks are the biggest *Big New Thing* for decades. They are now of central importance to all information technology. With the recent explosive growth of the internet, they are rapidly "she" becoming of crucial importance to all of modern society.

13.2. A Network Architecture Example: WWW

The World Wide Web is the *Big New Thing* in computer networking.

History:

In 1989, Tim Berners Lee proposed a global hypertext project, to be known as the World Wide Web. Based on the earlier "Enquire" work, it was designed to allow people to work together by combining their knowledge in a web of hypertext documents. Tim Berners Lee wrote the first World Wide Web server and the first client, a wysiwyg hypertext browser/editor which ran in the NeXTStep environment. This work was started in October 1990, and the program "World-WideWeb" was first made available within CERN in December, and on the Internet at large in the summer of 1991.

Through 1991 and 1993, Tim Berners Lee continued working on the design of the Web, coordinating feedback from users across the Internet. His initial specifications of URIs, HTTP and HTML were refined and discussed in larger circles as the Web technology spread.

See also:

A browser, or viewer program is used to fetch and display "pages" of information from a server. A page is simply an ASCII text file, written using a simple markup language called Hypertext Meta Language (HTML). You may find an introduction
Uniform Resource Locators - URLs

The URL is the basis of the WWW. Think of a URL as an address that can lead you to any file on any machine anywhere in the world. Unlike the common postal address, however, these are written backwards. (Actually backwards makes more sense. My postal address was:

   HP Bischof
   3002 ST RT 48
   Oswego, 13126 NY,
   USA.

But if you want to deliver a letter to me, shouldn’t you first go to the USA, then NY, then Oswego, then 3002 ST RT 48, then to HP Bischof? The URL is written in that more logical order.)

A URL defines the location of a WWW page in the following way:

   service:host:port/file and resource details

For example:

   http://www.cs.rit.edu:80/~hpb/CS3/all-2.2.html#section4
   http://www.av.digital.com/cgi-bin/query?pg=q&what=web

URLs on the Web don’t have to use the HTTP protocol. Some other URLs you might encounter are:

ftp file transfer protocol
news
   for Usenet news groups
telnet
   for telnet
mailto
   to send email to a specific address
Connection Establishment
To fetch a WWW page, the browser application process running on your local computer first establishes a connection to the remote host.

What this means is that the browser process uses the facilities of the network connecting the two computers to send a "connection request" message to a server process running on the computer whose name was given in the URL.

If the remote server process is prepared to accept the connection, it responds with a "connection accepted" message.

Note that we are, for the moment, ignoring the process of "looking up" the remote host - discovering the network address associated with its domain name.

The HTTP Protocol
Once the two application processes have an established connection between them, they can communicate reliably.

The browser then sends a request, in ordinary plain text, to the server, thus:

```
GET /home.html
```

The string GET something is one of many commands defined in the Hypertext Transfer Protocol, HTTP. The server responds by returning the contents of a file.

Finally, the browser process interprets the HTML markup in the returned file, and displays it to the user.
13.3. What is the Internet

The Internet (short for inter networking, the practice of linking technologically different and independently operated networks), is a network of networks which allows users to communicate using electronic mail, to retrieve data stored in databases, and to access distant computers. The "core" of Internet includes the National Science Foundation’s NSFNET, the Department of Energy’s Energy Science Network (ESnet), the NASA Science Internet (NSI) as well as Defense’s ARPANET and Terrestrial Wideband Network (TWBnet). Internet also includes a larger, and continually expanding, collection of interconnected regional, campus, and other networks throughout the U.S. and overseas, as well as several networks that provide service on a for-profit basis.

These linked networks are independently operated; there is no central control of Internet. Internet began as an Advanced Research Projects Agency research project to investigate computer networking technology. The networks that comprise the National Research and Education Network (NREN), a component of the High Performance Computing and Communications Program, are a part of the current Internet.

Copied from:

13.4. What is an Intranet

A local area network which may not be connected to the Internet, but which has some similar functions. Some organizations set up World Wide Web servers on their own internal networks so employees have access to the organization’s Web document.

Copied from:

13.5. Protocol

(From: Douglas Comer defines a protocol as "a formal description of message formats and the rules two or more machines must follow to exchange those messages."

Protocols usually exist in two forms. First, they exist in a textual form for humans to understand. Second, they exist as programming code for computers to understand. Both forms should ultimately specify the precise interpretation of every bit of every message exchanged across a network.

Protocols exist at every point where logical program flow crosses between hosts. In other words, we need protocols every time we want to do something on another computer. Every time we want to print something on a network printer we need protocols. Every time we want to download a file we need protocols. Every time we want to save our work on disk, we don’t need protocols - unless the disk is on a network file server.

Usually multiple protocols will be in use simultaneously. For one thing, computers usually do several things at once, and often for several people at once. Therefore, most protocols support multi-tasking. Also, one operation can involve several protocols. For example, consider the NFS (Network File System) protocol. A write to a file is done with an NFS operation, that uses another protocol (RPC) to perform a function call on a remote host, that uses another protocol (UDP) to deliver a data gram to a port on a remote host, that uses another protocol to deliver a datagram on an Ethernet, and so on. Along the way we made need to lookup host names (using the DNS protocol), convert data to a network standard form (using the XDR protocol), find a routing path to the host (using one or many of numerous protocols) - I think you get the idea.
13.6. Protocol Layers

Protocol layering is a common technique to simplify networking designs by dividing them into functional layers, and assigning protocols to perform each layer’s task.

For example, it is common to separate the functions of data delivery and connection management into separate layers, and therefore separate protocols. Thus, one protocol is designed to perform data delivery, and another protocol, layered above the first, performs connection management. The data delivery protocol is fairly simple and knows nothing of connection management. The connection management protocol is also fairly simple, since it doesn’t need to concern itself with data delivery.

Protocol layering produces simple protocols, each with a few well-defined tasks. These protocols can then be assembled into a useful whole. Individual protocols can also be removed or replaced.

The most important layered protocol designs are the Internet’s original DoD model, and the OSI Seven Layer Model. The modern Internet represents a fusion of both models.

•

•
13.7. The OSI Seven-Layer Model

(From: An Internet Encyclopedia) In the 1980s, the European-dominated International Standards Organization (ISO), began to develop its Open Systems Interconnection (OSI) networking suite. OSI has two major components: an abstract model of networking (the Basic Reference Model, or — seven-layer model —), and a set of concrete protocols. The standard documents that describe OSI are for sale and not currently available online.

Parts of OSI have influenced Internet protocol development, but none more than the abstract model itself, documented in OSI 7498 and its various addenda. In this model, a networking system is divided into layers. Within each layer, one or more entities implement its functionality. Each entity interacts directly only with the layer immediately beneath it, and provides facilities for use by the layer above it. Protocols enable an entity in one host to interact with a corresponding entity at the same layer in a remote host.

The seven layers of the OSI Basic Reference Model are (from bottom to top):

- **The Physical Layer** describes the physical properties of the various communications media, as well as the electrical properties and interpretation of the exchanged signals. This layer defines the size of Ethernet coaxial cable, the type of BNC connector used, and the termination method.

- **The Data Link Layer** describes the logical organization of data bits transmitted on a particular medium. Ex: this layer defines the framing, addressing and checksumming of Ethernet packets.

- **The Network Layer** describes how a series of exchanges over various data links can deliver data between any two nodes in a network. Ex: this layer defines the addressing and routing structure of the Internet.

- **The Transport Layer** describes the quality and nature of the data delivery. Ex: this layer defines if and how retransmissions will be used to ensure data delivery.

- **The Session Layer** describes the organization of data sequences larger than the packets handled by lower layers. Ex: this layer describes how request and reply packets are paired in a remote procedure call.

- **The Presentation Layer** describes the syntax of data being transferred. Ex: this layer describes how floating point numbers can be exchanged between hosts with different math formats.

- **The Application Layer** describes how real work actually gets done. Ex: this layer would implement file system operations.

The original Internet protocol specifications defined a four-layer model, and protocols designed around it (like TCP) have difficulty fitting neatly into the seven-layer model. Most newer designs use the seven-layer model.

13.8. TCP/IP

TCP/IP is the essential two-layer program that each Internet point-of-presence (POP) or SLIP/PPP user must use.

The **Transmission Control Protocol** (a protocol is a formal set of rules for communicating) manages the packaging of data into the packets that get routed on different paths over the Internet and reassembled at their destination.

The **Internet Protocol** handles the address part of each data packet so that it is routed to the right destination.

TCP/IP can be used on many data-link layers (can support many network hardware implementations).
These two protocols are the most important, TCP/IP is really a suite of protocols. (Some of these are viewed as alternative protocols and others as application protocols.) The ones you are most likely to use (directly or indirectly) are: HTTP, FTP, Telnet, Gopher, PPP, and SMTP.

Related protocols, some of them may be included in a TCP/IP package:

- User Datagram Protocol (UDP)
- Telnet
- File Transfer Protocol (FTP)
- Trivial File Transfer Protocol (TFTP)
- Simple Mail Transfer Protocol (SMTP)
- Gopher protocol
- Hypertext Transport Protocol (HTTP)
13.9. TCP/IP Layer

TCP/IP is normally considered to be a 4 layer system:

Link layer:
the network interface layer. Includes normally the device driver in an OS.

Network layer:
The network layer handles the movement of the packets around a network. Routing of packets takes place here. IP (Internet Protocol), ICMP (Internet Control Message Protocol), and IGMP (Internet Group Management Protocol) provide the network layer in the TCP/IP suite.
Transport Layer:
Provides a flow of data between two hosts for the application layer. There are two different protocols: TCP (Transmission Control Protocol) and UDP (User Datagram Protocol).

The function of the TCP protocol, is to provide a:

reliable all data is delivered correctly
connection-oriented the protocol provides procedures for establishing interprocess connections.
byte stream ie, no visible packetisation so far as the application processes are concerned
end-to-end
... interprocess communication service.

The User Datagram Protocol provides a connectionless alternative transport service to TCP for applications where reliable stream service is not needed. UDP datagrams can be dropped, duplicated or delivered out of order, same as for IP.

Application Layer:
The application layer handles the details of the particular application.

- telnet for remote login
- ftp the file transfer protocol
- SMTP simple mail transfer protocol
- SNMP simple network management protocol
An example:

.so Pic/16/use_tcp_ip.pic
13.10. Internet Addresses

Every Internet-connected system has a unique Internet host address. This is a 32 bit, or 4 byte, binary number. Internet addresses are written as a dotted sequence of the form:

\[ a.b.c.d \]

where \( a, b, c \) and \( d \) etc, are the decimal values (ranging from 0 to 255) of the 4 bytes which make up the internet address, for example:

129.3.20.4

129.3.20.4 is the IP address of ilon, or to use its full name

ilon.cs.oswego.edu

We will later see how the name of a computer is mapped to its IP-address.
13.11. IP Address Classes

boxht=0.5i boxwid=0.19i y_dist=0.8i line_ht=0.5 bit_border=0.1

define bits Y
  for i = 1 to 32 do X box with .w at last box.e if ( i % 8 == 0 ) then Z line from last box.ne + ( 0, bit_border) to last box.se - ( 0, bit_border) Z X Y

define tags Y
  # Class_box first second text
  line from $1.ne + ( $2 * boxwid, 0 ) to $1.ne + ( $2 * boxwid, line_ht )
  line from $1.ne + ( $3 * boxwid, 0 ) to $1.ne + ( $3 * boxwid, line_ht )
  box invis with .c at $1.ne + ( ( $3-$2 )/2+$2)*boxwid, boxht/2 ) $4 Y

A: box invis wid 1i "Class A" bits(A) A_1: box with .w at A.e + ( 0 * boxwid, 0 ) "0"
  tags(A, 1, 8, "7 bits — netid") tags(A, 8, 32, "24 bits — hostid")

B: box invis wid 1i "Class B" with .n at A.s - ( 0, y_dist ) bits(B) B_1: box with .w at B.e + ( 0 * boxwid, 0 ) "1" B_1: box with .w at B.e + ( 1 * boxwid, 0 ) "0"
  tags(B, 2, 16, "14 bits — netid") tags(B,16, 32, "16 bits — hostid")

C: box invis wid 1i "Class C" with .n at B.s - ( 0, y_dist ) bits(C) C_1: box with .w at C.e + ( 0 * boxwid, 0 ) "1" C_1: box with .w at C.e + ( 1 * boxwid, 0 ) "1" C_1: box with .w at C.e + ( 2 * boxwid, 0 ) "0"
  tags(C, 3, 24, "21 bits — netid") tags(C,24, 32, "8 bits — hostid")

D: box invis wid 1i "Class D" with .n at C.s - ( 0, y_dist ) bits(D) D_1: box with .w at D.e + ( 0 * boxwid, 0 ) "1" D_1: box with .w at D.e + ( 1 * boxwid, 0 ) "1" D_1: box with .w at D.e + ( 2 * boxwid, 0 ) "1" D_1: box with .w at D.e + ( 3 * boxwid, 0 ) "0"
  tags(D, 4, 32, "28 bits — multicast group id")

E: box invis wid 1i "Class E" with .n at D.s - ( 0, y_dist ) bits(E) E_1: box with .w at E.e + ( 0 * boxwid, 0 ) "1" E_1: box with .w at E.e + ( 1 * boxwid, 0 ) "1" E_1: box with .w at E.e + ( 2 * boxwid, 0 ) "1" E_1: box with .w at E.e + ( 3 * boxwid, 0 ) "1" E_1: box with .w at E.e + ( 4 * boxwid, 0 ) "0"
  tags(E, 5, 32, "27 bits — reserved for future use")

]
13.12. Ethernet Address
An IP-address only makes sense to the TCP/IP suite. A data link such as an Ethernet or a token ring has its own addressing scheme (often 48 bits).
The-byte address is often used, which is divided into a 3-byte vendor ID and a 3-byte vendor-defined field. Ethernet manufacturers are assigned a unique vendor ID, and are then responsible for insuring that all of their devices have unique addresses in the last 3 bytes.
A network, such as an Ethernet can be used by different network layers at the same time. See also

13.13. Encapsulation
When an application sends data using TCP is sent down the protocol stack.
A physical property of an Ethernet frame is, that the size of its data must be between 46 and 1500 bytes.

<table>
<thead>
<tr>
<th>Ethernet-header</th>
<th>IP-header</th>
<th>TCP-header</th>
<th>application-header</th>
<th>application data</th>
<th>Ethernet-trailer</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>20</td>
<td>20</td>
<td>46 to 1500 bytes</td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

13.14. TCP Ports
TCP is using protocol port numbers to identify the ultimate destination.
How does one determine the port to communicate with?
— Well known ports
— Randomly assigned ports.

See

13.15. Socket
Two popular Application Programming Interface using TCP/IP protocols are called sockets and TLI (transport layer interface).
A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program. The java.net package provides two classes--Socket and ServerSocket--that implement the client side of the connection and the server side of the connection, respectively.
A socket is a network communications endpoint.
A socket is an object from which messages are sent and received.
Socket operations resemble file operations in many respects:
• Data transfer operations on sockets work just like read and write operations on files.
• A socket is closed, just like a file, when communications is finished.

See also:

13.16. java.net
Through the classes in java.net, Java programs can use TCP or UDP to communicate over the Internet. The URL, URLConnection, Socket, and ServerSocket classes all use TCP to communicate over the network. The DatagramPacket, DatagramSocket, and MulticastSocket classes are for use with UDP.

13.17. Getting Information

Host info

```java
import java.net.*;
import java.io.*;
import java.util.*;
public class HostInfo {
    public static void main(String argv[]) {
        InetAddress ipAddr;
        try {
            ipAddr = InetAddress.getLocalHost();
            System.out.println("This is "+ipAddr);
        } catch (UnknownHostException e) {
            System.out.println("Unknown host");
        }
    }
}
```

Source Code: Src/16/HostInfo.java

```
% java HostInfo
This is yps/129.21.38.198
```

13.18. Daytime Client

see /etc/services.

```java
import java.net.*;
import java.io.*;
import java.util.*;
public class DayTime {
    private void printMessage() {
        System.out.println("-h ----> help");
        System.out.println("[-host hostName]");
        System.out.println("[-port port]");
    }
    /**
    * Parse the command line arguments and sets variables.
    */
```
public class DayTimeServer extends Thread {

    public static void main(String argv[]) {
        DayTime aDayTime = new DayTime();
        aDayTime.parseArgs(argv);
        aDayTime.doTheJob();
    }
}

Source Code: Src/16/DayTime.java

% java DayTime
host: spiegel.cs.rit.edu
port: 13
java.net.ConnectException: Operation timed out
Wed Nov 20 17:59:31 EST 2013

13.19. Daytime Server
ServerSocket aServerSocket;
int port = 4242;

public DayTimeServer() {
}

public DayTimeServer(int port) {
    try {
        aServerSocket = new ServerSocket(port);
        System.out.println("Listening on port: " + aServerSocket.getLocalPort());
    } catch (Exception e) {
        System.out.println(e);
    }
}

private void printMessage() {
    System.out.println("-h ----> help");
    System.out.println(" -port port");
    System.out.println(" {-port port}");
    System.out.println("or ");
    System.out.println(" no argument");
}

/**
 * Parse the command line arguments and sets variables.
 */
private void parseArgs(String args[]) {
    for (int i = 0; i < args.length; i++) {
        if (args[i].equals("-h"))
            printMessage();
        else if (args[i].equals("-port")) {
            port = new Integer(args[++i]).intValue();
            new DayTimeServer(port).start();
        }
    }
}

public void run() {
    try {
        for (;;) {
            Socket clnt = aServerSocket.accept();
            System.out.println(clnt.toString());
            PrintWriter out = new PrintWriter(clnt.getOutputStream(), true);
            out.println("It is now: "+ new Date());
            clnt.close();
        }
    } catch (Exception e) {
        System.out.println(e);
        e.printStackTrace();
    }
}
61 public static void main(String argv[]) {  
62 if (argv.length == 0) 
63 new DayTimeServer(0).start(); 
64 else 
65 new DayTimeServer().parseArgs(argv); 
66 }
67
Source Code: Src/16/DayTimeServer.java

% java DayTimeServer -port 12345 &
% java DayTime -port 12345
host: spiegel.cs.rit.edu
port: 12345
Socket[addr=/129.21.36.56,port=64586,localport=12345]
It is now: Wed Nov 20 18:05:28 EST 2013

13.20. Reading from and Writing to a Socket

1 import java.io.*;
2 import java.net.*;
3
4 class HpEchoSocketTest {
5 
6 Socket s;
7 String hostName;
8 int port;
9 PrintWriter out = null;
10 BufferedReader in = null;
11
12 public HpEchoSocketTest() { 
13 }
14 public HpEchoSocketTest(String name, int port) {
15     hostName = name;
16     this.port = port;
17 }
18
19 public void parseArgs(String args[]) {
20     for (int i = 0; i < args.length; i++) {
21         if (args[i].equals("-host"))
22             hostName = args[++i];
23         else if (args[i].equals("-port"))
24             port = new Integer(args[++i]).intValue();
25     }
26 }
27
28 public void createIOconections() throws Exception {
29     try {
30         s = new Socket(hostName, port);
31         out = new PrintWriter( s.getOutputStream(), true);
32         in = new BufferedReader( new InputStreamReader( s.getInputStream()));
33     }
34     }
35 }
36
Source Code: Src/16/HpEchoSocketTest.java

% java HpEchoSocketTest -host spiegel.cs.rit.edu -port 12345 &
host: spiegel.cs.rit.edu
port: 12345
Socket[addr=/129.21.36.56,port=64586,localport=12345]
It is now: Wed Nov 20 18:05:28 EST 2013
```java
} catch (Exception e) {
    System.out.println(e.toString());
    System.exit(1);
}

public void closeIOconections() throws Exception {
    try {
        in.close();
        out.close();
    } catch (Exception e) {
        System.out.println(e.toString());
        System.exit(1);
    }
}

public void readAndPrint() throws Exception {
    InputStream ins;
    OutputStream os;

    BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
    String userInput;

    while ((userInput = stdIn.readLine()) != null) {
        out.println(userInput);
        System.out.println("echo: " + in.readLine());
    }
    stdIn.close();
}

public void doTheJob() {
    try {
        System.out.println("host: " + hostName);
        System.out.println("port: " + port);
        HpEchoSocketTest aHpEchoSocketTest = new HpEchoSocketTest();
        createIOconections();
        readAndPrint();
        closeIOconections();
    } catch (Exception e) {
        System.out.println(e);
    }
}

public static void main(String[] args) {
    HpEchoSocketTest st;
    String host = "spiegel.cs.rit.edu";
    int port = 12345;
    HpEchoSocketTest aHpEchoSocketTest = new HpEchoSocketTest();
    aHpEchoSocketTest.parseArgs(args);
    aHpEchoSocketTest.doTheJob();
}

Source Code: Src/16/HpEchoSocketTest.java
```
import java.net.*;
import java.io.*;
import java.util.*;

/**
 * Java Echo Server
 *
 * @param args
 * @return
 */
public class EchoServer extends Thread {
    ServerSocket aServerSocket;
    int port = 4242;

    public EchoServer() {
    }

    public EchoServer(int port) {
        try {
            aServerSocket = new ServerSocket(port);
            System.out.println ("Listening on port: "+ aServerSocket.getLocalPort());
        } catch (Exception e) {
            System.out.println(e);
        }
    }

    private void printMessage() {
        System.out.println(" -h ----> help");
        System.out.println(" -port port");
        System.out.println(" {-port port}");
        System.out.println(" or ");
        System.out.println(" no argument");
    }

    private void parseArgs(String args[]) {
        for (int i = 0; i < args.length; i++) {
            if (args[i].equals("-h"))
                printMessage();
            else if (args[i].equals("-port")) {
                port = new Integer(args[++i]).intValue();
                new EchoServer(port).start();
            }
        }
    }

    public void run() {
        try {
            for (;;) {
                Socket clnt = aServerSocket.accept();
                System.out.println(clnt.toString());
                PrintWriter out = new PrintWriter (clnt.getOutputStream (), true);
                BufferedReader in = new BufferedReader( new InputStreamReader( clnt.getInputStream()));
                while (true) {
                    String input = in.readLine();
                    out.println(input);
                    out.flush();
                }
            }
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
out.println("back: " + input);

    } catch(Exception e) {
        System.out.println(e);
        e.printStackTrace();
    }

    public static void main(String argv[]) {
        if (argv.length == 0 )
            new EchoServer(12345).start();
        else
            new EchoServer().parseArgs(argv);
    }

Source Code: Src/16/EchoServer.java

% java EchoServer
Listening on port: 12345
% java HpEchoSocketTest -port 12345
host: null
port: 12345
jkjkj
echo: back: jkjkj
lkklk
echo: back: lkklk

These are the typical steps:
1. Open a socket.
2. Open an input stream and output stream to the socket.
3. Read from and write to the stream according to the server’s protocol.
4. Close the streams.
5. Close the socket.

13.21. Multi Client Server and Client

• telnet allows many clients to connect
• Server:

    // java MTS

    import java.net.*;
    import java.io.*;
    import java.util.*;

    public class MTS extends Thread {


ServerSocket listen;
static String hostName = "yps";
int port = 4242;
int id = 0;

public MTS() {
    listen = null;
}

public MTS(int port) {
    try {
        listen = new ServerSocket(port);
        System.out.println("Listening on port: " + getLocalPort());
    } catch(Exception e) {
        System.out.println(e);
    }
}

public MTS(int port, int id) {
    this(port);
    this.id = id;
}

public int getLocalPort () {
    return listen.getLocalPort();
}

private void printMessage() {
    System.out.println("-h ----> help");
    System.out.println("[-host hostName");
    System.out.println(" -port port");
    System.out.println(" {-port port}");
    System.out.println(" or ");
    System.out.println(" no argument");
}

/**
 * Parse the commandline arguments and sets variables.
 */
public void parseArgs(String args[]) {
    for (int i = 0; i < args.length; i++) {
        if (args[i].equals("-h"))
            printMessage();
        else if (args[i].equals("-host"))
            hostName = args[++i];
        else if (args[i].equals("-port")) {
            port = new Integer(args[++i]).intValue();
            new MTS(port).listenToPort();
        }
    }
}

public void run() {

try {
    Socket clnt = listen.accept();
    System.out.println(clnt.toString());
    PrintWriter out = new PrintWriter(clnt.getOutputStream(), true);
    out.println("It is now: " + new Date());
    System.out.println(id + " .... falling asleep");
    sleep(1000);
    System.out.println("\t" + id + " .... wake up");
    listen.close();
} catch(Exception e) {
    System.out.println(e);
    e.printStackTrace();
}

public void listenToPort() {
    try {
        int id = 0;
        for(;;) {
            Socket clnt = listen.accept();
            System.out.println("Somebody connected ... ");
            MTS aServer = new MTS(0, id++);
            aServer.start();
            System.out.println("offer ... " +
                               aServer.getLocalPort());
            PrintWriter out = new PrintWriter(clnt.getOutputStream(), true);
            out.println(aServer.getLocalPort());
            // clnt.close();
        }
    } catch(Exception e) {
        System.out.println(e);
        e.printStackTrace();
    }
}

public static void main(String argv[]) {
    if ( argv.length == 0 )
        new MTS(0).listenToPort();
    else
        new MTS().parseArgs(argv);
}

Source Code: Src/16/MTS.java

• Client:
// java MTSclient -host spiegel -port 50405
import java.net.*;
import java.io.*;
import java.util.*;
public class MTSclient {
    String hostName = "yps";
    int   port = 13;

    private void printMessage() {
        System.out.println("-h ----> help");
        System.out.println("[-host hostName]");
        System.out.println("[-port port]");
    }

    /**
     * Parse the commandlind arguments and sets variables.
     */
    public void parseArgs(String args[]) {
        for (int i = 0; i < args.length; i ++) {
            if (args[i].equals("-h"))
                printMessage();
            else if (args[i].equals("-host"))
                hostName = args[++i];
            else if (args[i].equals("-port"))
                port = new Integer(args[++i]).intValue();
        }
    }

    public void doTheJob() {
        try {
            Socket socket = new Socket(hostName, port);
            BufferedReader din = new BufferedReader (new InputStreamReader (socket.getInputStream()));
            String newPort = din.readLine ();
            System.out.println ("Use from now in port: " + newPort);
            socket.close();
            din.close();

            socket = new Socket(hostName, new Integer(newPort).intValue());
            din = new BufferedReader (new InputStreamReader (socket.getInputStream()));
            System.out.println("got: " + din.readLine () );
        } catch (Exception e) {
            System.out.println (e);
        }
    }

    public static void main(String argv[]) {  
        MTSclient aMTSclient = new MTSclient();
        aMTSclient.parseArgs(argv);
Source Code: Src/16/MTSclient.java

Result:

# Window 1:
% java MTS
Listening on port: 1234
Somebody connected ...
local port: 53064
Sleep for a while

# Window 2:
java MTSclient -port 1234 &
[1] 6500
yps 7 142 local port: 1234
Mon Oct 24 09:05:55 EDT 2005

13.22. Connection to an URL

Class URL represents a Uniform Resource Locator, a pointer to a "resource" on the World Wide Web. A resource can be something as simple as a file or a directory, or it can be a reference to a more complicated object, such as a query to a database or to a search engine. More information on the types of URLs and their formats can be found at:

```java
import java.io.*;
import java.net.URL;
import java.net.MalformedURLException;

public class Url_Read {
    public static void readFromUrl(String theUrl) {
        URL aUrl = null;
        BufferedReader in = null;
        String line;

        try {
            aUrl = new URL(theUrl);
            System.out.println("getPort() " + aUrl.getPort());
            System.out.println("getHost() " + aUrl.getHost());
            System.out.println("getProtocol() " + aUrl.getProtocol());
            System.out.println("getFile() " + aUrl.getFile());
            System.out.println("getRef() " + aUrl.getRef());

            in = new BufferedReader(
                new InputStreamReader( aUrl.openStream() ) );
        }
    }
}
```
while ((line = in.readLine()) != null) {
    System.out.println(line);
}

in.close();

} catch (MalformedURLException e) {
    System.err.println("Something is wrong with this " +
    theUrl + ".");
    System.exit(1);
} catch (IOException e) {
    System.err.println("Couldn’t get I/O for the connection to: " +
    theUrl);
    System.exit(1);
}

public static void main(String args[]) {
    if (args.length != 1) {
        System.err.println("Usage: java Url_Read url");
        System.exit(1);
    }

    try {
        readFromUrl(args[0]);
    }
    catch (NumberFormatException e) {
        System.out.println(args[0] + " is not a number ;-(-");
    }
}

Source Code: Src/16/Url_Read.java

% java Url_Read http://www.cs.rit.edu/~hpb | sed 15q
getPort() -1
getHost() www.cs.rit.edu
getProtocol() http
getFile() ~/hp
getRef() null

<HTML>

<HEAD>
<title>Hans-Peter Bischof’s Home Page</title>
</HEAD>

<FRAMESET cols="230,*">
    <frame name="toc" TARGET="_main" src="toc.html" scrolling="auto">
    <frame name="intro" src="intro.html" scrolling="auto">
13.23. Datagram Socket

- A datagram socket is the sending or receiving point for a packet delivery service.
- Each packet sent or received on a datagram socket is individually addressed and routed.
- Multiple packets sent from one machine to another may be routed differently, and may arrive in any order.
- UDP broadcasts sends and receives are always enabled on a DatagramSocket.

An example:

- Server:

```java
import java.net.*;
import java.io.*;
import java.util.*;

public class DayTimeUDPServer extends Thread {
    public void run() {
        DatagramSocket socket = null;
        try {
            socket = new DatagramSocket(port);
            System.out.println("Listening on port: "+ socket.getLocalPort());
        } catch (Exception e) {
            System.out.println(e);
        }
    }

    private void printMessage() {
        System.out.println("-h ----> help");
        System.out.println("[-host hostName]");
        System.out.println(" -port port");
        System.out.println(" {-port port}");
        System.out.println(" or ");
        System.out.println(" no argument");
    }

    public void parseArgs(String args[]) {
        for (int i = 0; i < args.length; i++) {
            if (args[i].equals("-h")) {
                printMessage();
            } else if (args[i].equals("-host")) {
            }
        }
    }
}
```
hostName = args[++i];
        else if (args[i].equals("-port")) {
            port = new Integer(args[++i]).intValue();
            new DayTimeUDPServer(port).start();
        }
    }
}

public void run() {
    byte[] buf = new byte[256];
    try {
        for(;;) {
            String sendThis = "es schlaegt: " + new Date();
            DatagramPacket packet = new DatagramPacket(buf, buf.length);
            socket.receive(packet);
            InetAddress address = packet.getAddress();
            int port = packet.getPort();
            buf = sendThis.getBytes();
            packet = new DatagramPacket(buf, buf.length, address, port);
            System.out.println("Sending to port: " + port);
            System.out.println("Sending data: " + new String(buf));
            socket.send(packet);
        }
    } catch(Exception e) {
        System.out.println(e);
        e.printStackTrace();
    }
}

public static void main(String argv[]) {
    if (argv.length == 0)
        new DayTimeUDPServer(0).start();
    else
        new DayTimeUDPServer().parseArgs(argv);
}

Source Code: Src/16/DayTimeUDPServer.java

Client:

import java.net.*;
import java.io.*;
import java.util.*;
public class DayTimeUDP {
    String hostName = "yps";
    int    port = 13;

    private void printMessage() {
        System.out.println("-h ----> help");
        System.out.println("[-host hostName]");
        System.out.println("[-port port]");
    }
}
/**
 * Parse the command line arguments and sets variables.
 */

public void parseArgs(String args[]) {
    for (int i = 0; i < args.length; i++) {
        if (args[i].equals("-h"))
            printMessage();
        else if (args[i].equals("-host"))
            hostName = args[++i];
        else if (args[i].equals("-port"))
            port = new Integer(args[++i]).intValue();
    }
}

public void doTheJob() {
    try {
        byte buf[] = new byte[64];
        InetAddress aInetAddress = InetAddress.getByName(hostName);
        DatagramPacket dp = new DatagramPacket(buf, buf.length);
        DatagramSocket socket = new DatagramSocket();
        DatagramPacket packet = new DatagramPacket(buf,
            buf.length, aInetAddress, port);
        socket.send(packet);

        System.out.println("host: "+hostName);
        System.out.println("port: "+port);
        System.out.println("after creation");
        socket.receive(dp);
        System.out.println("received: -"+
            new String(dp.getData())+"-");
        socket.close();
    } catch (Exception e) {
        System.out.println(e);
        e.printStackTrace();
    }
}

public static void main(String argv[]) {
    DayTimeUDP aDayTimeUDP = new DayTimeUDP();
    aDayTimeUDP.parseArgs(argv);
    aDayTimeUDP.doTheJob();
}

Source Code: Src/16/DayTimeUDP.java

• Execution:
# Window 1:
% java DayTimeUDPServer -port 53818
Listening on port: 53818
Sending to port: 53840
Mon Oct 24 09:05:56 EDT 2005

# Window 2:
% java DayTimeUDP -port 53818
host: yps
port: 53818
after creation
received: -es schlaegt: Mon Oct 24 09:05:56 EDT 2005
13.24. Remote Method Invocation

See also
Part of the text and programs are from there. Copyright belongs to and
See also

The Java Remote Method Invocation (RMI) system allows an object running in one Java Virtual Machine (VM) to invoke methods on an object running in another Java VM. RMI provides for remote communication between programs written in the Java programming language.

Distributed object applications need to:

Locate remote objects
Applications can use one of two mechanisms to obtain references to remote objects. An application can register its remote objects with RMI’s simple naming facility, the rmiregistry, or the application can pass and return remote object references as part of its normal operation.

Communicate with remote objects
Details of communication between remote objects are handled by RMI; to the programmer, remote communication looks like a standard Java method invocation.
Load class bytecodes for objects that are passed as parameters or return values

Because RMI allows a caller to pass pure Java objects to remote objects, RMI provides the necessary mechanisms for loading an object’s code as well as transmitting its data.

The illustration below depicts an RMI distributed application that uses the registry to obtain references to a remote object. The server calls the registry to associate a name with a remote object. The client looks up the remote object by its name in the server’s registry and then invokes a method on it.

The problem is: you need a reference of an object, before you can send a method to it.


13.27. SenderProxy/Receiver Proxy.png

13.28. Stubs

```java
public final class HelloImplementation_Stub
    extends java.rmi.server.RemoteStub
    implements HelloInterface, java.rmi.Remote
{
    private static final long serialVersionUID = 2;

    private static java.lang.reflect.Method $method_sayHello_0;

    static {
        try {
            $method_sayHello_0 = HelloInterface.class.getMethod("sayHello", new java.lang.Class[] {});
        } catch (java.lang.NoSuchMethodException e) {
            throw new java.lang.NoSuchMethodError("stub class initialization failed");
        }
    }

    // constructors
    public HelloImplementation_Stub(java.rmi.server.RemoteRef ref) {
        super(ref);
    }

    // methods from remote interfaces
    // implementation of sayHello()
    public java.lang.String sayHello() throws java.rmi.RemoteException
    {
        try {
            Object $result = ref.invoke(this, $method_sayHello_0, null, 6043973830760146143L);
            return ((java.lang.String) $result);
        } catch (java.lang.RuntimeException e) {
            throw e;
        } catch (java.rmi.RemoteException e) {
            throw e;
        } catch (java.lang.Exception e) {
            throw new java.rmi.UnexpectedException("undeclared checked exception", e);
        }
    }
}
```
Source Code: Src/16_D/HelloImplementation_Stub.java

Note: rmic -keep can generate the stubs.
13.29. Remote Method Innovation Registry

Where does the sender proxy come from?
13.30. Passing Non-remote Objects

A non-remote object, that is passed as a parameter of a remote method invocation or returned as a result of a remote method invocation, is passed by copy; that is, the object is serialized using the Java Object Serialization mechanism.

So, when a non-remote object is passed as an argument or return value in a remote method invocation, the content of the non-remote object is copied before invoking the call on the remote object.

When a non-remote object is returned from a remote method invocation, a new object is created in the calling virtual machine.
13.31. Advantages of Dynamic Code Loading

- download the bytecodes (or simply code) of an object’s class if the class is not defined in the receiver’s virtual machine.
- types and the behavior of an object, previously available only in a single virtual machine, can be transmitted to another, possibly remote, virtual machine.
- RMI passes objects by their true type, so the behavior of those objects is not changed when they are sent to another virtual machine.
- allows new types to be introduced into a remote virtual machine, thus extending the behavior of an application dynamically.
13.32. Remote Interfaces, Objects, and Methods

- Distributed application built using Java RMI is made up of interfaces and classes.
- In a distributed application some of the implementations are assumed to reside in different virtual machines.
- Objects that have methods that can be called across virtual machines are remote objects.
- An object becomes remote by implementing a remote interface, which has the following characteristics.
  - A remote interface extends the interface java/rmi.Remote.
  - Each method of the interface declares java/rmi.RemoteException in its throws clause, in addition to any application-specific exceptions.
- RMI passes a remote stub for a remote object.
- The stub acts as the local representative, or proxy, for the remote object and basically is, to the caller, the remote reference.
- The caller invokes a method on the local stub, which is responsible for carrying out the method call on the remote object.
- A stub for a remote object implements the same set of remote interfaces that the remote object implements.
- This allows a stub to be cast to any of the interfaces that the remote object implements.
- This also means that only those methods defined in a remote interface are available to be called in the receiving virtual machine.
13.33. RMI & JDK 1.5

RMI has been enhanced in the following areas:

- **Dynamic Generation of Stub Classes** - This release adds support for the dynamic generation of stub classes at runtime, obviating the need to use the Java Remote Method Invocation (Java RMI) stub compiler, rmic, to pre-generate stub classes for remote objects.
  
  Note that rmic must still be used to pre-generate stub classes for remote objects that need to support clients running on earlier versions.

- **Standard SSL/TLS Socket Factory Classes** - This release adds standard Java RMI socket factory classes, `javax.rmi.ssl.SslRMIClientSocketFactory` and `javax.rmi.ssl.SslRMIServerSocketFactory`, which communicate over the Secure Sockets Layer (SSL) or Transport Layer Security (TLS) protocols using the Java Secure Socket Extension (JSSE).

- **Launching rmid or a Java RMI Server from inetd/xinetd** - A new feature, provided by the `System.inheritedChannel` method, allows an application to obtain a channel (`java.nio.channels.SocketChannel` or `java.nio.channels.ServerSocketChannel`, for example) inherited from the process that launched the virtual machine (VM).
  
  Such an inherited channel can be used to either service a single incoming connection (as with `SocketChannel`) or accept multiple incoming connections (as with `ServerSocketChannel`). Therefore, Java networking applications launched by `inetd` (Solaris(tm) Operating System) or `xinetd` (Linux) can now obtain the `SocketChannel` or `ServerSocketChannel` inherited from `inetd/xinetd`.

Stolen from:

13.34. Creating Distributed Applications Using RMI

1. Design and implement the components of your distributed application.
2. Compile sources and generate stubs.
3. Make classes network accessible.
4. Start the application.

**Compile Sources and Generate Stubs**

Compile as usual ... see `makefile`

**Make Classes Network Accessible**

In this step you make everything—the class files associated with the remote interfaces, stubs, and other classes that need to be downloaded to clients.

**Start the Application**

Starting the application includes running the RMI remote object registry, the server, and the client.
13.35. **Intro Example**

We have to design a protocol that allows jobs to be submitted to the server and results of the job to be returned to the client. This protocol is expressed in interfaces supported by the server and by the objects that are submitted to the server, as shown in the following figure.

```
box with .sw at (1.00,8.62) width 1.25 height 0.75 box with .sw at (3.75,8.62) width 1.25 height 0.75 line -> from 2.250,9.125 to 3.750,9.125 line from 3.750,8.750 to 3.750,8.750 line -> from 3.750,8.875 to 2.250,8.875 "Client" at 1.250,8.914 ljust "Server" at 4.000,8.914 ljust "submit task" at 2.375,9.289 ljust "return results" at 2.375,8.539 ljust
```
13.36. Hello World

The Client:

```java
import java.rmi.*;

public class HelloC {
    public static void main(String args[]) {
        String message = "";
        try {
            HelloInterface obj = (HelloInterface)Naming.lookup("//spiegel/IamAHelloImplementationObject");
            message = obj.sayHello();
            System.out.println(message);
        } catch (Exception e) {
            System.out.println("HelloC exception: "+e.getMessage());
            e.printStackTrace();
        }
    }
}

Source Code: Src/16_D/HelloC.java
```
The interface:

1 public interface HelloInterface extends java.rmi.Remote {
2     String sayHello() throws java.rmi.RemoteException;
3 }

Source Code: Src/16_D/HelloInterface.java
The implementation of the interface:

```java
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class HelloImplementation
    extends UnicastRemoteObject
    implements HelloInterface {

    public HelloImplementation() throws RemoteException {
    }

    public String sayHello() throws RemoteException {
        return "Hello World my Friend!";
    }
}
```

Source Code: Src/16_D/HelloImplementation.java
The implementation of the server:

```java
import java.rmi.*;

public class HelloServer {

    public static void main(String args[])
    {
        // System.setSecurityManager(new RMISecurityManager());

        try {
            HelloInterface obj = new HelloImplementation();
            Naming.rebind("/spiegel/IamAhelloImplementationObject", obj);
            System.out.println("HelloServer bound in registry");
        } catch (Exception e) {
            System.out.println("HelloImpl err: " + e.getMessage());
            e.printStackTrace();
        }
    }

    Source Code: Src/16_D/HelloServer.java

Compilation

all:
    javac HelloInterface.java
    javac HelloImplementation.java
    javac HelloC.java HelloServer.java
    rmiregistry &
    sleep 1
    java HelloServer &
    sleep 4
    java HelloC

The stub:

Source Code: Src/16_D/Keep_HelloImplementation_Stub.java
```
RMISecurityManager

The class defines a default security policy for RMI applications (not applets). For code loaded from a class loader, the security manager disables all functions except class definition and access. This class may be subclassed to implement a different policy. To set a RMISecurityManager, add the following to an application’s main() method:

```
System.setSecurityManager(new RMISecurityManager());
```

If no security manager has been set, RMI will only load classes from local system files as defined by CLASSPATH.

Naming

is the bootstrap mechanism for obtaining references to remote objects based on Uniform Resource Locator (URL) syntax. The URL for a remote object is specified using the usual host, port and name:

```
rmi://host:port/name
```

- **host** = host
  - name of registry (defaults to the current host)
- **port** = port
  - number of registry (defaults to the registry port number)
- **name** = name
  - for remote object

The makefile:

Source Code: Src/16_D/makefile
% make -f Makefile
javac HelloInterface.java
javac HelloImplementation.java
# rmic HelloImplementation
javac HelloC.java HelloServer.java
rmiregistry &
sleep 1
java HelloServer &
sleep 4
HelloServer bound in registry
java HelloC
Hello World my friend.
Note: Make sure that rmiregistry is dead before you log out!

Note: Make sure that every java server is dead before you log out!

```
#!/bin/sh

TO_FIND="registry"
if [ $# -eq 1 ]
then
  if [ $1 -eq "java" ]
  then
    TO_FIND="java"
  fi
fi

ps -edf | grep $TO_FIND | grep -v grep | grep -v kill | grep -v vi | awk '{print $2}'
while read x
do
  echo "kill -9 $x"
  kill -9 $x 2> /dev/null
done
exit 0
```

Source Code: Src/16_D/killIt

% ps -edf | grep java
The improved makefile:

Source Code: Src/16_D/makefile
% make
rmic HelloImpl
javac HelloC.java
rmiregistry &
sleep 1
java HelloImpl &
sleep 4
    HelloImpl: HelloImpl(String s)
HelloServer bound in registry
java HelloC
Hello World my friend.
killIt java
kill -9 26491
kill -9 26489
kill -9 26438
kill -9 26332
kill -9 26501
Killed
killIt
Killed
The registry by default runs on port 1099. To start the registry on a different port, specify the port number in the command. For example, to start the registry on port 2001:

```
% rmiregistry 2001
```

For example, if the registry is running on port 2001 in the Hello World example, here is the call required to bind HelloServer to the remote object reference:

```
Naming.lookup("//yps:2001/HelloServer", obj);
```
13.37. Hello World II

The Client:

```java
import java.rmi.*;

public class HelloC {
    public static void main(String args[]) {
        String message = "";
        try {
            Hello obj =
                (Hello)Naming.lookup("//yps:2001/HelloServer");
            message = obj.sayHello();
            System.out.println(message);
        } catch (Exception e) {
            System.out.println("Something went wrong: "+
                e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Source Code: Src/16_P/HelloC.java
The Server:

```java
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class HelloImpl
extends UnicastRemoteObject
implements Hello
{
private String name;

public HelloImpl(String s) throws RemoteException {
    name = s;
}

public String sayHello() throws RemoteException {
    return "Stanley Kubrick was there!";
}

public static void main(String args[])
{
    // Create and install a security manager
    // System.setSecurityManager(new RMISecurityManager());
    try {
        HelloImpl obj = new HelloImpl("HelloServer");
        Naming.rebind("//yps:2001/HelloServer", obj);
        System.out.println("HelloServer bound in registry");
    } catch (Exception e) {
        System.out.println("HelloImpl err: " + e.getMessage());
        e.printStackTrace();
    }
}
```

Source Code: Src/16_P/HelloImpl.java
The interface:

```java
public interface Hello extends java.rmi.Remote {
    String sayHello() throws java.rmi.RemoteException;
}
```

Source Code: Src/16_P/Hello.java
The makefile:

```
1 2 3  all: Hello.class HelloC.class HelloImpl.class \ 4    HelloImpl_Skel.class HelloImpl_Stub.class
5  
6    rmiregistry 2001 &
7    sleep 1
8    java HelloImpl &
9    sleep 4
10   java HelloC
11   killIt java
12   killIt
13
14  HelloImpl_Skel.class HelloImpl_Stub.class: HelloImpl.java
15    rmic HelloImpl
16
17  Hello.class:   Hello.java
18    javac Hello.java
19
20  HelloC.class:  HelloC.java
21    javac HelloC.java
22
23  HelloImpl.class: HelloImpl.java
24    javac HelloImpl.java
25
26  clean:
27    rm -f *class
```

Source Code: Src/16_P/makefile
13.38. Multiple Servers

Client:

```java
import java.rmi.*;

public class Client {
    public static void main(String args[]) {
        String message = "";
        try {
            MyServer obj =
                (MyServer) Naming.lookup("//yps:2001/Server1");
            message = obj.sayHello();
            System.out.println(message);

            obj = (MyServer) Naming.lookup("//yps:2001/Server2");
            message = obj.sayHello();
            System.out.println(message);
        } catch (Exception e) {
            System.out.println("Something went wrong: " +
                              e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Source Code: Src/16_M/Client.java
Server 1:

```java
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class Server1Impl
extends UnicastRemoteObject
implements MyServer
{
    private String name;

    public Server1Impl(String s) throws RemoteException {
        name = s;
    }

    public String sayHello() throws RemoteException {
        return "Server1()";
    }

    public static void main(String args[]) {
        try {
            Server1Impl obj = new Server1Impl("Server1");
            Naming.rebind("//yps:2001/Server1", obj);
        } catch (Exception e) {
            System.out.println("Server1Impl err: "+ e.getMessage());
            e.printStackTrace();
        }
    }

    Source Code: Src/16_M/Server1Impl.java
```
Server 2:

```java
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class Server2Impl
    extends UnicastRemoteObject
    implements MyServer
{
    private String name;

    public Server2Impl(String s) throws RemoteException {
        name = s;
    }

    public String sayHello() throws RemoteException {
        return "Server2()";
    }

    public static void main(String args[]) {
        // Create and install a security manager
        // System.setSecurityManager(new RMISecurityManager());
        try {
            Server2Impl obj = new Server2Impl("Server2");
            Naming.rebind("//yps:2001/Server2", obj);
            System.out.println("Server2Server bound in registry");
        } catch (Exception e) {
            System.out.println("Server2Impl err: " + e.getMessage());
            e.printStackTrace();
        }
    }
}

Source Code: Src/16_M/Server2Impl.java

% make
rmic Server1Impl
rmic Server2Impl
javac Client.java
rmiregistry 2001 &
sleep 1
java Server1Impl &
java Server2Impl &
sleep 4
Server2Server bound in registry
Server1 bound in registry
java Client
Server1()
Server2()
```
13.39. Running Multiple Server on different Machines

- Before a client can connect to a server, a server process must run on this machine.
- Server can be started — on a UNIX box — during
  - boot time
  - via `inetd`
  - "by hand"

- In order to start a server by hand, you have to log on this machine.
- `ssh`
- `rsh` ????

13.40. Startup Multiple Server on different Machines

Client:

```java
import java.rmi.*;
import java.math.*;

public class Client {

    public static void doIt(String catServer, String mouseServer, int port) {

        MyServer aCatServer;
        MyServer aMouseServer;
        Point aPoint = new Point(4, 2);

        System.out.println("In Client: cat is on: " + catServer);
        System.out.println("In Client: mouse is on: " + mouseServer);
        System.out.println("In Client: port is: " + port);
        try {
            aCatServer = (MyServer)Naming.lookup("rmi://" +
                catServer + ":" + port + "/CatServer");

            aMouseServer = (MyServer)Naming.lookup("rmi://" +
                mouseServer + ":" + port + "/MouseServer");

            // -------------- Cat --------------------
            System.out.println("In Client: aCatServer.movePoint(aPoint): " +
                (aPoint = aCatServer.movePoint(aPoint)).toString());
            System.out.println("In Client: aCatServer.movePoint(aPoint): " +
                aCatServer.movePoint(aPoint).toString());
            System.out.println("In Client: aCatServer.movePoint(aPoint): " +
                aCatServer.movePoint(aPoint).toString());

            // -------------- Mouse -------------------
            System.out.println("In Client: aMouseServer.movePoint(aPoint): " +
```
(aPoint = aMouseServer.movePoint(aPoint)).toString() );
System.out.println("In Client: aMouseServer.movePoint(aPoint): " +
aMouseServer.movePoint(aPoint).toString() );
System.out.println("In Client: aMouseServer.movePoint(aPoint): " +
aMouseServer.movePoint(aPoint).toString() );

} catch (Exception e) {
    System.out.println("Something went wrong: " +
        e.getMessage());
    e.printStackTrace();
}

public static void main(String args[]) {
    int port = 1099;
    String catServer = "yps";
    String mouseServer = "yps";

    if (args.length >= 1) {
        catServer = args[0];
    }
    if (args.length >= 2) {
        mouseServer = args[1];
    }
    if (args.length == 3) {
        try {
            port = Integer.parseInt(args[2]);
        } catch (NumberFormatException e) {
            System.out.println("Hm, port = " +
                args[2] + " is not valid.");
            System.exit(1);
        }
    }

    if (args.length > 3) {
        System.out.println("Usage: " +
            "java Client [CatServer [MouseServer [port]]]");
        System.exit(1);
    }

    doIt(catServer, mouseServer, port);
}

Source Code: Src//16_MS/Client.java
Interface:

```java
public interface MyServer extends java.rmi.Remote {
    Point movePoint(Point aPoint) throws java.rmi.RemoteException;
    int getX() throws java.rmi.RemoteException;
    int getY() throws java.rmi.RemoteException;
}
```

Source Code: Src//16_MS/MyServer.java
public class CatServer extends UnicastRemoteObject implements MyServer
{
    final private int DELTA = 10;
    private int x;
    private int y;
    private Point aPoint;

    public CatServer() throws RemoteException {
    }

    public Point movePoint(Point aPoint) throws RemoteException {
        System.out.println("\tIN CatServer: movePoint(): "+ aPoint.toString());
        return aPoint.move(DELTA, DELTA);
    }

    public int getX() throws RemoteException {
        System.out.println("\t\tIN CatServer: getX(): "+ x);
        return x;
    }

    public int getY() throws RemoteException {
        System.out.println("\t\tIN CatServer: getY(): "+ y);
        return x;
    }

    public static void main(String args[])
    {
        int port = 1099;

        // System.setSecurityManager(new RMISecurityManager());

        if (args.length == 1)
        try{
            port = Integer.parseInt(args[0]);
        }
        catch (NumberFormatException e) {
            System.out.println("Hm, port = " +
            args[0] + " is not valid.");
            System.exit(1);
        }

        try {
            CatServer obj = new CatServer();
            System.out.println("\tIN CatServer: "+
            "rmi://:" + port + "/CatServer");
            Naming.rebind("rmi://:" + port + "/CatServer", obj);
System.out.println("\tIN CatServer bound in registry");

} catch (RemoteException e) {
    System.out.println("CatServer RemoteException ");
    e.printStackTrace();
}

} catch (Exception e) {
    System.out.println("CatServer err: "+ e.getMessage());
    e.printStackTrace();
}

Source Code: Src//16_MS/CatServer.java
/**
 * This class implements a point in a two dimensional area.
 * All methods print the method name, when they are called.
 * state information includes:
 *
 * @version $Id$
 *
 * RIT's home page: <a href="http://www.cs.rit.edu/~hpbl">RIT</a>
 *
 * Revisions:
 * $Log$
 */
import java.io.*;

public class Point implements Serializable {
    private int x; // x coordinate of the point
    private int y; // y coordinate of the point

    /**
     * Constructor.
     * initialize x and y values of a point
     *
     * @param x x coordinate
     * @param y y coordinate
     *
     * @return a Point object
     */
    public Point(int _x, int _y){
        this.x = _x;
        this.y = _y;
    }

    private void writeObject(ObjectOutputStream s) throws IOException {
        s.defaultWriteObject();
    }

    private void readObject(ObjectInputStream s) throws IOException {
        try {
            s.defaultReadObject();
        } catch (ClassCastException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }  

    /**
     * initialzes x and y of a point.
     *
* @param x int x coordinate
* @param y int y coordinate
* @return a Point object
*/
public Point initPoint(int _x, int _y) {
    this.x = _x;
    this.y = _y;
    return this;
}

/**
* moves a point
*
* @param _x int delta x value
* @param _y int delta y value
* @return a Point object
*/
public Point move(int _x, int _y) {
    this.x += _x;
    this.y += _y;
    return this;
}

/**
* Returns the x coordinate of a point
*
* @return x value
*/
public int getX() {
    return this.x;
}

/**
* Returns the y coordinate of a point
*
* @return y value
*/
public int getY() {
    return this.y;
}

/**
* Returns a String representation of the point
*
* @return String representation of the point
*/
public String toString() {
    return "Point at (" + x + "/" + y + ")";
}
06 }

Source Code: Src/16_MS/Point.java
Makefile:

```
all: Point.class \
    CatServer_Skel.class CatServer_Stub.class \
    MouseServer_Skel.class MouseServer_Stub.class \
    MyServer.class Client.class

    fireItUp

CatServer_Skel.class CatServer_Stub.class: CatServer.java
    rmic CatServer

MouseServer_Skel.class MouseServer_Stub.class: MouseServer.java
    rmic MouseServer

MyServer.class: MyServer.java
    javac MyServer.java

Client.class: Client.java
    javac Client.java

CatServer.class: CatServer.java
    javac CatServer.java

MouseServer.class: MouseServer.java
    javac MouseServer.java

Point.class: Point.java
    javac Point.java

clean:
    rm -f *class
```

Source Code: Src/16_MS/makefile
Result:

IN CatServer: //yps:2001/CatServer
IN MouseServer: /yps:2001/MouseServer
IN CatServer bound in registry
In Client: cat is on: yps
In Client: mouse is on: yps
In Client: port is: 2001
IN MouseServer bound in registry
IN CatServer: movePoint(): Point at (4/2)
In Client: aCatServer.movePoint(aPoint): Point at (14/12)
IN CatServer: movePoint(): Point at (14/12)
In Client: aCatServer.movePoint(aPoint): Point at (24/22)
IN CatServer: movePoint(): Point at (14/12)
In Client: aCatServer.movePoint(aPoint): Point at (24/22)
Start of a fireItUp Script:

```bash
#!/bin/sh

KILL_IT="killIt; killIt java"
ME="`who am i | sed 's/ .*//`"
HOSTNAME="`hostname`"
USEDHOSTS="yps yps yps" # <-- modify here ...
WD=`pwd`

remote_cmd() # bg host cmd
{
    echo "$HOSTNAME $ME" > $HOME/.rhosts
    if [ $1 = "bg" ]
    then
        rsh $2 "rm -f $HOME/.rhosts; cd $WD && $3" &
    else
        rsh $2 "rm -f $HOME/.rhosts; cd $WD && $3"
    fi
}

kill_all()
{
    for i in $USEDHOSTS
    do
        remote_cmd fg $i "$KILL_IT" 2>&1 > /dev/null
    done
}

kill_all
sleep 2

echo 1
rmiregistry &
echo "Waiting for rmiregistry .... chrr ... "; sleep 1
java CatServer &

echo 2
remote_cmd bg yps "rmiregistry &"
```
echo 3
echo "Waiting for rmiregistry .... chrr ... "; sleep 2
remote_cmd bg yps "java MouseServer 6"

echo 4
echo "Waiting for the servers .... chrr ... "; sleep 2
remote_cmd fg yps "java Client $HOSTNAME stones"

kill_all
exit 0

Source Code: Src/16_MS/fireItUp
13.41. Calculating PI

```bash
% make
javac Client.java
rmiregistry &
sleep 1
java PiServer &
sleep 4
    PiServer: PiServer()
PiServer bound in registry
java Client
3.1415926536
% java Client 100
3.141592653589793238462643383279502884197
16939937510582097494459230781640628620899
86280348253421170680
%```

Server Interface:

```java
tenish
1  public interface MyServer extends java.rmi.Remote {
2      String sayHello() throws java.rmi.RemoteException;
3  }
```

Source Code: Src/16_M/MyServer.java

**Class java/gmath.BigDecimal**

Immutable, arbitrary-precision signed decimal numbers. A BigDecimal consists of an arbitrary precision integer value and a non-negative integer scale, which represents the number of decimal digits to the right of the decimal point. (The number represented by the BigDecimal is intVal/7**scale.) BigDecimals provide operations for basic arithmetic, scale manipulation, comparison, format conversion and hashing.
The compute engine, a remote object in the server, takes tasks from clients, runs them, and returns any results. The tasks are run on the machine where the server is running. This sort of distributed application could allow a number of client machines to make use of a particularly powerful machine or one that has specialized hardware.

Client:

```java
import java.rmi.*;
import java.math.*;

public class Client {
    public static void doIt(String host, String port, int digits) {
        String message = "";
        try {
            MyServer obj = (MyServer)Naming.lookup("//" +
                    host + ":" + port + "/PiServer");
            System.out.println(obj.computePi(digits));
        } catch (Exception e) {
            System.out.println("Something went wrong: " +
                    e.getMessage());
            e.printStackTrace();
        }
    }

    public static void main(String args[]) {
        int digits = 10;
        String host = "yps";
        String port = "";

        if (args.length >= 1) {
            try {
                digits = Integer.parseInt(args[0]);
            } catch (NumberFormatException e) {
                System.out.println("Hm , digits = " + args[0]);
                System.exit(1);
            }
        }

        if (args.length >= 2) {
            host = args[1];
        }

        if (args.length == 3) {
            try {
                port = args[2];
                Integer.parseInt(port);
            } catch (NumberFormatException e) {
                System.out.println("Port = " + port + " is not valid.");
            }
        }
    }
}
```
48             System.exit(1);
49         }
50
51     }
52     if ( args.length > 3 ) {
53         System.out.println("Usage: java Client [digits [host [port]]]");
54         System.exit(1);
55     }
56     doIt(host, port, digits);
57 }
58 }

Source Code: Src/16_C/Client.java
Interface:

```java
import java.math.*;

public interface MyServer extends java.rmi.Remote {
    BigDecimal computePi(int digits)
    throws java.rmi.RemoteException;
}
```

Source Code: Src/16_C/MyServer.java
```java
import java.rmi.*;
import java.math.*;
import java.rmi.server.UnicastRemoteObject;

public class PiServer
    extends UnicastRemoteObject
    implements MyServer
{
    /**
     * Construct a task to calculate pi to the specified
     * precision.
     */
    public PiServer() throws RemoteException {
        System.out.println("PiServer: PiServer()");
    }

    /**
     * Compute the value of pi to the specified number of
     * digits after the decimal point. The value is
     * computed using Machin’s formula:
     * pi/4 = 4*arctan(1/5) - arctan(1/239)
     * and a power series expansion of arctan(x) to
     * sufficient precision.
     */
    public BigDecimal computePi(int digits) throws RemoteException {
        int scale = digits + 5;
        BigDecimal arctan1_5 = arctan(5, scale);
        BigDecimal arctan1_239 = arctan(239, scale);
        BigDecimal pi = arctan1_5.multiply(FOUR).subtract(
            arctan1_239).multiply(FOUR);
        return pi.setScale(digits,
            BigDecimal.ROUND_HALF_UP);
    }
}
```
* Compute the value, in radians, of the arctangent of
* the inverse of the supplied integer to the specified
* number of digits after the decimal point. The value
* is computed using the power series expansion for the
* arc tangent:
*    * arctan(x) = x - (x^3)/3 + (x^5)/5 - (x^7)/7 +
*    * (x^9)/9 ...
* /
public static BigDecimal arctan(int inverseX,
       int scale)
{
  BigDecimal result, numer, term;
  BigDecimal invX = BigDecimal.valueOf(inverseX);
  BigDecimal invX2 =
       BigDecimal.valueOf(inverseX * inverseX);
  numer = ONE.divide(invX, scale, roundingMode);
  result = numer;
  int i = 1;
  do {
    numer =
       numer.divide(invX2, scale, roundingMode);
    int denom = 2 * i + 1;
    term =
       numer.divide(BigDecimal.valueOf(denom),
       scale, roundingMode);
    if ((i % 2) != 0) {
      result = result.subtract(term);
    } else {
      result = result.add(term);
    }
    i++;
  } while (term.compareTo(ZERO) != 0);
  return result;
}

public static void main(String args[])
{
  // Create and install a security manager
  // System.setSecurityManager(new RMISecurityManager());
  try {
    PiServer obj = new PiServer();
    Naming.rebind("//yps:2042/PiServer", obj);
    System.out.println("PiServer bound in registry");
  } catch (Exception e) {
    System.out.println("PiServer err: " + e.getMessage());
    e.printStackTrace();
  }
}

Source Code: Src/16_C/PiServer.java
13.42. Receiving and Sending Objects

Interface:

```java
import java.util.*;

public interface HashTableInterface extends java.rmi.Remote {
    Hashtable playWithAHashTable(String t)
        throws java.rmi.RemoteException;
}
```

Source Code: Src/16_Hash/HashTableInterface.java
Client:

```java
import java.rmi.*;
import java.util.*;

public class HashTableC {
    public static void main(String args[]) {
        String plusMovie = "Smoke Signals";
        Hashtable aHashTable = new Hashtable();
        aHashTable.put("plusplus Movie", "Comedian Harmonists");

        System.out.println("Client: aHashTable local = " +
                           aHashTable.toString());

        try {
            HashtableInterface obj =
                (HashtableInterface)Naming.lookup("//yps/HelloServer" +
                                           aHashTable.toString());

            try {
                aHashTable = obj.playWithAHashTable(plusMovie);
            } catch (Exception e) {
                System.out.println("HelloApplet exception: " +
                                   e.getMessage());
                e.printStackTrace();
                System.out.println("Client: aHashTable remote = " +
                                   aHashTable.toString());
            }
        }
    }
}
```

Source Code: Src/16_Hash/HashTableC.java
import java.util.*;
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class HashTableServer
    extends UnicastRemoteObject
    implements HashTableInterface {
    private String name;

    public HashTableServer(String s) throws RemoteException {
        System.out.println("HashTableServer: HashTableServer(String s)\n                   name = s;\n    }

    public Hashtable playWithAHashTable(String t)
        throws java.rmi.RemoteException {
        Hashtable aHashTable = new Hashtable();
        aHashTable.put("plusplus Movie", t);
        System.out.println("server: aHashTable = 
                   aHashTable.toString());
        t = "done";
        return aHashTable;
    }

    public static void main(String args[])
    {
        // System.setSecurityManager(new RMISecurityManager());
        try {
            HashTableServer obj = new HashTableServer("HelloServer");
            Naming.rebind("//yps/HelloServer", obj);
            System.out.println("HelloServer bound in registry");
            } catch (Exception e) {
            System.out.println("HashTableServer err: " + e.getMessage() +
                   e.printStackTrace();
            }
    }

Source Code: Src/16_Hash/HashTableServer.java
Result:

% make
rmic HashTableServer
javac HashTableC.java
rmiregistry &
sleep 1
java HashTableServer &
sleep 4
   HashTableServer: HashTableServer(String s)
HelloServer bound in registry
java HashTableC
Client: aHashTable = {plusplus Movie=Comedian Harmonists}
   server: aHashTable = {plusplus Movie=Smoke Signals}
Client: aHashTable = {plusplus Movie=Smoke Signals}
Client: plusMovie = Smoke Signals
killIt java
kill -9 27386
Killed
killIt
kill -9 27366
kill -9 27374
# make clean

13.43. RMI and Multi Threaded Systems

How does a remote method gets executed?

The Interface:

```java
1    public interface MultiTInterface extends java.rmi.Remote {
2    String comeBackASAP() throws java.rmi.RemoteException;
3    String sleepForAwhile() throws java.rmi.RemoteException;
4    }
```

Source Code: Src/16_T/MultiTInterface.java

The Client:

```java
1    import java.rmi.*;
2
3    public class MultiTC {
4       public static void main(String args[]) {
5       String message = "";
6       System.out.println("Going ... ");
7       try {
8          MultiTInterface obj =
9             (MultiTInterface)Naming.lookup("//spiegel:2001/MultiTServer");
10         if ( args.length == 0 ) {
11            System.out.println("Call sleepForAwhile ...");
12            message = obj.sleepForAwhile();
13            System.out.println(message);
14         } else {
15            System.out.println("Call comeBackASAP ...");
```
message = obj.comeBackASAP();
System.out.println(message);
}

} catch (Exception e) {
    System.out.println("Something went wrong: " +
    e.getMessage());
e.printStackTrace();
}

}

public static void main(String args[]) {
    try {
        MultiTImpl obj = new MultiTImpl("MultiTServer");
        Naming.rebind("/spiegel:2001/MultiTServer", obj);
        System.out.println("MultiTServer bound in registry");
    } catch (Exception e) {
        System.out.println("MultiTImpl err: " + e.getMessage())
e.printStackTrace();
    }
}
Source Code: Src/16_T/MultiTImpl.java

The makefile:

```makefile
all: MultiTInterface.class MultiTC.class MultiTImpl.class MultiTImpl_Skel.class MultiTImpl_Stub.class
     rmiregistry 2001 &
      sleep 1
      java MultiTImpl &
      sleep 4
      java MultiTC 1 &
      java MultiTC &
      killIt java
      killIt

MultiTImpl_Skel.class MultiTImpl_Stub.class: MultiTImpl.java
     rmic MultiTImpl

MultiTInterface.class: MultiTInterface.java
     javac MultiTInterface.java

MultiTC.class: MultiTC.java
     javac MultiTC.java

MultiTImpl.class: MultiTImpl.java
     javac MultiTImpl.java

clean:
     rm -f *class
```

Source Code: Src/16_T/makefile

### 13.44. Dynamic Class Loading

RMI allows parameters, return values and exceptions passed in RMI calls to be any object that is serializable. RMI uses the object serialization mechanism to transmit data from one virtual machine to another and also annotates the call stream with the appropriate location information so that the class definition files can be loaded at the receiver.

When parameters and return values for a remote method invocation are unmarshaled to become live objects in the receiving VM, class definitions are required for all of the types of objects in the stream. The unmarshaling process first attempts to resolve classes by name in its local class loading context (the context class loader of the current thread). RMI also provides a facility for dynamically loading the class definitions for the actual types of objects passed as parameters and return values for remote method invocations from network locations specified by the transmitting endpoint. This includes the dynamic downloading of remote stub classes corresponding to particular remote object implementation classes (and used to contain remote references) as well as any other type that is passed by value in RMI calls, such as the subclass of a declared parameter type, that is not already available in the class loading context of the unmarshaling side.

To support dynamic class loading, the RMI runtime uses special subclasses of `java/io.ObjectOutputStream` and `java/io.ObjectInputStream` for the marshal streams that it uses for marshaling and unmarshaling RMI parameters and return values.
13.45. Java Object Serialization Security Issues

See here:

Problem:

- Classes/Objects are to remote process (JVM)
- Unmarshalled
- Developer:
  - Trust communication channel
  - Assume binary objects can not be changed
  - Assume Serialization is safe
- Idea: how to allow "xxx" to login, instead of "hpb"

```java
import java.io.*;
import java.util.Date;

public class ObjectWriter_5 {
    public static void main( String args[] ) {
        try {
            FileOutputStream ostream = new FileOutputStream("object_5.data");
            ObjectOutputStream p = new ObjectOutputStream(ostream);
            p.writeObject("User: " + "hpb");
            p.close();
        }
        catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Source Code: Src/9_was/ObjectWriter_5.java

% od -c object_5.data
00000000 254 355 005 t U s e r : h p b
00000020
% od -c x.data
00000000 254 355 005 t U s e r :
00000015
% echo -n xxx >> x.data
% od -c x.data
00000000 254 355 005 t U s e r : x x x
00000020

- the same can be done with classes, and objects

Arbitrary Code Execution

- Code Reuse attack (return-oriented programming)
  - control of the call stack
  - Executed carefully chosen machine instructions (gadgets)
if (1 + 2 + payload + 16 > s->s3->rrec.length) return 0; /* silently discard payload */

Restrict Deserialization

- Default ObjectInputStream will deserialize any serializable class
- Class Black/White Listening

14. Collections

40 questions:

14.1. What is a Collection

Stolen from

- A is simply an object that groups multiple elements into a single unit.
- Collections are used to store, retrieve and manipulate data, and to transmit data from one method to another.
- The primary use of the Collection interface is to pass around collections of objects where maximum generality is desired.
- Collections typically represent data items that form a natural group.

Sorting:

```java
import java.util.*;

public class Sort {
    public static void main(String args[]) {
        List l = Arrays.asList(args);
        Collections.sort(l);
        System.out.println(l);
    }
}
```

Source Code: Src/9/Sort.java
14.2. How could we Implement the Previous Example?

- Assume the list is an array
- We handle only String objects
- How about this:

```java
public class BubbleSort {

    public static void printIt(String aCollection[]) {
        for (int index=0; index<aCollection.length; index++)
            System.out.println(index + "\t" + aCollection[index]);
    }

    public static void sort(String aCollection[]) {
        for (int index=0; index < aCollection.length - 1; index++) {
            for (int walker=0; walker < aCollection.length - 1; walker++) {
                if (aCollection[walker].compareTo(aCollection[walker+1]) > 0)
                    String tmp = aCollection[walker];
                    aCollection[walker] = aCollection[walker+1];
                    aCollection[walker+1] = tmp;
            }
        }
    }

    public static void main( String args[]) {
        String[] aCollection = new String[3];
        aCollection[0] = "c";
        aCollection[1] = "b";
        aCollection[2] = "a";
        sort(aCollection);
        printIt(aCollection);
    }
}
```

Source Code: Src/9/BubbleSort.java

- What do we need:
  - a way to access every object in the array
  - we use the String objects compareTo method
  - Will this work for other kind of objects?
import java.util.*;

class HpCollections {
    public static void sort(List aList) {
        Object anArray[] = null;
        for (int index=0; index<anArray.length - 1; index++) {
            for (int walker=0; walker<anArray.length - index - 1; walker++) {
                String left = (String) anArray[walker];
                String right = (String) anArray[walker+1];
                if (left.compareTo(right) > 0) {
                    Object tmp = anArray[walker];
                    anArray[walker] = anArray[walker + 1];
                    anArray[walker+1] = tmp;
                }
            }
        }  
        aList = Arrays.asList(anArray);
    }
    public String toString() {
        String s = new String();
        for (Object o: anArray)
            s = s + "/" + o;
        return s;
    }
    public static void main(String args[]) {
        args = new String[4];
        args[0] = "z"; args[1] = "x";
        List l = Arrays.asList(args);
        HpCollections.sort(l);
        System.out.println(l);
    }
}

Source Code: Src/Collection_5/HpCollections.java

import java.util.*;

class HpCollections_remove {
    public static void sort(List aList) {
        Object anArray[] = aList.toArray();
    }
    public String toString() {
        String s = new String();
        for (Object o: anArray)
            s = s + "/" + o;
        return s;
    }
    public static void main(String args[]) {
        args = new String[4];
        args[0] = "z"; args[1] = "x";
        List l = Arrays.asList(args);
        // HpCollections_remove.sort(l);
        // HpOKCollections.sort(l);
        // Collections.sort(l);
        HpCollections.sort(l);
        System.out.println(l);
    }
}

Source Code: Src/Collection_5/HpCollections_remove.java
for (int index=0; index<anArray.length - 1; index++) {
    for (int walker=0; walker<anArray.length - index - 1; walker++) {
        Comparable left = (Comparable) anArray[walker];
        Comparable right = (Comparable) anArray[walker+1];
        if (left.compareTo(right) > 0) {
            Object tmp = anArray[walker];
            anArray[walker] = anArray[walker + 1];
            anArray[walker + 1] = tmp;
        }
    }
}

for (Object o: anArray) {
    System.out.println("anArray: " + o);
    for (int index=0; index<anArray.length; index++) {
        aList.remove(index);
        aList.add(anArray[index]);
    }
}

Source Code: Src/Collection_5/HpCollections_remove.java

import java.util.*;

public class HpOKCollections {
    public static void sort(List list) {
        Object a[] = list.toArray();
        Arrays.sort(a); /////// this is the trick
        ListIterator i = list.listIterator();
        for (int j=0; j<a.length; j++) {
            i.next(); // this is it
            i.set(a[j]); // modification of the list
        }
    }
}

Source Code: Src/Collection_5/HpOKCollections.java

import java.util.*;

public class Sort {
    public static void main(String args[]) {
        String[] args = new String[4];
        args[0] = "z"; args[1] = "x";
        List l = Arrays.asList(args);
        // Collections.sort(l);
10     HpOKCollections.sort(l);
11     System.out.println(l);
12     }
13     }
14
    Source Code: Src/Collection_5/Sort.java

Why does it not work?
- take a look at this class

14.4. What Is a Collections Framework?
A collections framework is a unified architecture for representing and manipulating collections. All collections frameworks contain three things:

- Interfaces: abstract data types representing collections. Interfaces allow collections to be manipulated independently of the details of their representation.
- Implementations: concrete implementations of the collection interfaces. In essence, these are reusable data structures.
- Algorithms: methods that perform useful computations, like searching and sorting, on objects that implement collection interfaces. These algorithms are said to be polymorphic because the same method can be used on many different implementations of the appropriate collections interface. In essence, algorithms are reusable functionality.

14.5. Iterators

- The object returned by the method is very similar to an Enumeration, but differs in two respects:
  — Iterator allows the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
  — Method names have been improved.

Use of an iterator:

1
2          import java.util.*;
3
4          public class UseIterator {
5              public static void main(String args[]) {
6                  int index = 0;
7                  List l = Arrays.asList(args);
8                  Iterator aIterator = l.iterator();
9                  while ( aIterator.hasNext() ) {
10                      System.out.println(++index + " : "+
11                          (String)aIterator.next() );
12                  }
13              }
14          }

    Source Code: Src/9/UseIterator.java

• It reduces programming effort
• It increases program speed and quality
• It allows interoperability among unrelated APIs
• It reduces effort to design new APIs
• It fosters software reuse

14.7. Collection Interface

• The core interfaces is:

copied from URL http://java.sun.com/docs/books/tutorial/collections/interfaces/index.html

• The interface is the root of the collection hierarchy. A Collection represents a group of objects, known as its elements. Some Collection implementations
  — allow duplicate elements and others do not.
  — some are ordered and others unordered.

Collection is used to pass collections around and manipulate them when maximum generality is desired.
• Interfaces:
  Collection
  • Enumeration
  • List
  • Map
  • Queue
  • RandomAccess
  • Set
  • SortedMap
  • SortedSet

14.8. Set Interface

A is a collection that cannot contain duplicate elements.
• Two Set objects are equal if they contain the same elements.
• See also and
• A is an ordered collection

14.9. Lists

• Lists can contain duplicate elements.
14.10. Maps

- A is an object that maps keys to values.
- Maps cannot contain duplicate keys: Each key maps to one value.

14.11. Maps vs Collections

- Collections: add, remove, lookup
- Maps: key value pair, access values stored by key

14.12. A Picture
14.13. See Here
14.14. General Purpose Implementations

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Hash Table</th>
<th>Resizable Array</th>
<th>Balanced Tree</th>
<th>Linked List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Set</td>
<td>HashSet</td>
<td></td>
<td>TreeSet</td>
<td></td>
</tr>
<tr>
<td>Interface List</td>
<td>ArrayList</td>
<td></td>
<td></td>
<td>LinkedList</td>
</tr>
<tr>
<td>Interface Map</td>
<td>HashMap</td>
<td></td>
<td>TreeMap</td>
<td></td>
</tr>
</tbody>
</table>

- The fact that the new implementations are unsynchronized represents a break with the past
- If you need a synchronized collection, the synchronization wrappers, allow any collection to be transformed into a synchronized collection. Thus, synchronization is optional for the new collection implementations where it was mandatory for the old.
- As a rule of thumb, you should be thinking about the interfaces rather than the implementations.

14.15. Implementations: Set

- HashSet and
- TreeSet.
- HashSet is much faster (constant time vs. log time for most operations), but offers no ordering guarantees.
- If it is needed to use the operations in the SortedSet, or in-order iteration is important to use TreeSet.

14.16. Implementations: List

- ArrayList and
- LinkedList.
- ArrayList offers constant time positional access

14.17. Implementations: Map

- HashMap and
- TreeMap
- The situation for Map is exactly analogous to Set.

14.18. Algorithms

- prints out its arguments in lexicographic order:

```
1 import java.util.*;
2 public class Sort {
3     public static void main(String args[]) {
4         List l = Arrays.asList(args);
5         Collections.sort(l);
6         System.out.println(l);
7     }
8 }
9 ```
14.19. Examples: HashSet

```java
import java.util.HashSet;
import java.util.Set;

public class HashSetEx_1 {
    private Set<Integer> universe;
    private Set<Integer> fill(int soMany) {
        Set<Integer> universe = new HashSet<Integer>();
        for (int index = 0; index < soMany; index++)
            universe.add(new Integer(9999999 * index));
        return universe;
    }
    public static void main(String args[]) {
        Set<Integer> universe = null;
        HashSetEx_1 aHashSetEx_1 = new HashSetEx_1();
        universe = aHashSetEx_1.fill(253);
        System.out.println("1: " + universe);
        universe.remove(new Integer(1)); // universe.remove(new Integer(10));
        System.out.println("2: " + universe);
        universe.remove(new Integer(10));
        System.out.println("3: " + universe);
    }
}
```

Source Code: Src/9/HashSetEx_1.java

14.20. Examples: HashMap I

```java
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import java.util.Iterator;

public class HashMapEx_1 {
    private Map<String, String> universe;
    private Map<String, String> fill() {
        Map<String, String> universe = new HashMap<String, String>();
        for (int index = 0; index < 100; index++)
            universe.put("name", "index");
        return universe;
    }
    public static void main(String args[]) {
        Map<String, String> universe = null;
        HashMapEx_1 aHashMapEx_1 = new HashMapEx_1();
        universe = aHashMapEx_1.fill();
        System.out.println("1: " + universe);
        universe.remove("name");
        System.out.println("2: " + universe);
        universe.remove("name");
        System.out.println("3: " + universe);
    }
}
```

Source Code: Src/9/HashMapEx_1.java
public class HashMapEx {

    private Map<Integer, String> universe;

    private Map<Integer, String> fill(int soMany) {
        universe = new HashMap<Integer, String>();
        for (int index = 0; index < soMany; index++)
            universe.put(new Integer(index), "_" + index);
        return universe;
    }

    private Map<Integer, String> delete(int what) {
        try {
            for (Integer id : universe.keySet()) {
                System.out.println("try to delete: " + id);
                if (id.equals(new Integer(what)))
                    universe.remove(id);
                System.out.println("deleted: " + id);
            }
        } catch (Exception e) {
            System.out.println("Exception ..... ");
            e.printStackTrace();
        }
        return universe;
    }

    private Map<Integer, String> deleteUsingKeySetCorrect(int what) {
        try {
            Iterator aIterator = universe.keySet().iterator();
            while (aIterator.hasNext()) {
                aIterator.next();
                aIterator.remove();
            }
        } catch (Exception e) {
            System.out.println("Exception ");
            e.printStackTrace();
        }
        return universe;
    }

    public static void main(String args[]) {
        Map<Integer, String> universe;
        HashMapEx aHashMapEx = new HashMapEx();
        universe = aHashMapEx.fill(3);
    }
}
System.out.println("1: " + universe);
aHashMapEx.deleteUsingKeySetCorrect(1);

universe = aHashMapEx.fill(3);
aHashMapEx.delete(1);
System.out.println("2: " + universe);

Source Code: Src/9/HashMapEx.java

1: {0=_0, 1=_1, 2=_2}
0
1 Exception ..... 
java.util.ConcurrentModificationException
   at java.util.HashMap$HashIterator.nextEntry(HashMap.java:793)
   at java.util.HashMap$KeyIterator.next(HashMap.java:828)
   at HashMapEx.delete(HashMapEx.java:18)
   at HashMapEx.main(HashMapEx.java:37)
2: {0=_0, 2=_2}

From: Note that this implementation is not synchronized. If multiple threads access this map concurrently, and at least one of the threads modifies the map structurally, it must be synchronized externally. (A structural modification is any operation that adds or deletes one or more mappings; merely changing the value associated with a key that an instance already contains is not a structural modification.) This is typically accomplished by synchronizing on some object that naturally encapsulates the map. If no such object exists, the map should be “wrapped” using the Collections.synchronizedMap method. This is best done at creation time, to prevent accidental unsynchronized access to the map:

Map m = Collections.synchronizedMap(new HashMap(...));

The iterators returned by all of this class’s "collection view methods" are fail-fast: if the map is structurally modified at any time after the iterator is created, in any way except through the iterator’s own remove or add methods, the iterator will throw a ConcurrentModificationException. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, nondeterministic behavior at an undetermined time in the future.

Note that the fail-fast behavior of an iterator cannot be guaranteed as it is, generally speaking, impossible to make any hard guarantees in the presence of unsynchronized concurrent modification. Fail-fast iterators throw ConcurrentModificationException on a best-effort basis. Therefore, it would be wrong to write a program that depended on this exception for its correctness: the fail-fast behavior of iterators should be used only to detect bugs.

14.21. List Iterator

• An iterator for lists allows the programmer to traverse the list
  — in either direction
  — modify the list during iteration
  — obtain the iterator’s current position in the list.
• A ListIterator has no current element
• its cursor position always lies between the element that would be returned by a call to previous() and the element that would be returned by a call to next()
• In a list of length n, there are n+1 valid index values, from 0 to n, inclusive.

```java
import java.util.Stack;
import java.util.ListIterator;
import java.util.Collection;

public class ListItereatorEx {
    private Collection<String> fill(String words[]) {
        Stack<String> palindrom = new Stack<String>();
        for (String id : words) {
            palindrom.push(id);
        }
        return palindrom;
    }

    private Collection<String> leftRight() {
        ListIterator<String> aListIterator = palindrom.listIterator(2);
        String s = aListIterator.next();
        System.out.println("s = " + s);
        aListIterator.set("ZZ top");
        return palindrom;
    }

    public static void main(String args[]) {
        Collection<String> aStack;
        String theOnes[] = { "a", "b", "c", "d"};
        ListItereatorEx o = new ListItereatorEx();

        aStack = o.fill(theOnes);
        System.out.println("1: " + aStack);
        aStack = o.leftRight();
        System.out.println("2: " + aStack);
    }
}
```

Source Code: Src/9/ListItereatorEx.java

14.22. Collections.sort()
How does Collections.sort() work?
public class XX extends X {
    public XX() { System.out.println("XX()"); }
    public XX(int b) { System.out.println("XX(int b)"); }

    public static void main(String args[]) {
        X aX = new X();
        System.out.println("--------------------------");
        aX = new X(3);
        System.out.println("--------------------------");
        XX aXX = new XX();
        System.out.println("--------------------------");
        aXX = new XX(3);
        System.out.println("--------------------------");
        // aX = new XX(3);
        // aXX = new X(3);
    }
}

Source Code: Src/9/XX.java

From java doc

sort

public static void sort(List list)
Sorts the specified list into ascending order, according to the natural ordering of its elements. This sort is guaranteed to be stable: equal elements will not be reordered as a result of the sort.

The specified list must be modifiable, but need not be resizable.

The sorting algorithm is a modified mergesort (in which the merge is omitted if the highest element in the low sublist is greater than or equal to the lowest element in the high sublist). This algorithm offers guaranteed n log(n) performance. This implementation dumps the specified list.

Parameters:
list - the list to be sorted.

Throws:
ClassCastException - if the list contains elements that are not mutually comparable (for example, strings and integers).
UnsupportedOperationException - if the specified list’s list-iterator does not support the set operation.

See Also:
Comparable

14.23. Object Ordering

• A List l may be sorted as follows:
  Collections.sort(l);

<table>
<thead>
<tr>
<th>Class</th>
<th>Natural Ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>signed numerical</td>
</tr>
<tr>
<td>Character</td>
<td>unsigned numerical</td>
</tr>
<tr>
<td>Long</td>
<td>signed numerical</td>
</tr>
</tbody>
</table>
The Comparable interface consists of a single method:

```java
public interface Comparable {
    public int compareTo(Object o);
}
```

Example:

```java
import java.util.*;

public class Name implements Comparable {
    protected String firstName, lastName;

    public Name(String firstName, String lastName) {
        if (firstName==null || lastName==null)
            throw new NullPointerException();
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String firstName() {
        return firstName;
    }

    public String lastName() {
        return lastName;
    }

    public boolean equals(Object o) {
        if (!(o instanceof Name))
            return false;
        Name n = (Name)o;
        return n.firstName.equals(firstName) &&
               n.lastName.equals(lastName);
    }

    public String toString() {
        return firstName + " " + lastName;
    }

    public int compareTo(Object o) {
        Name n = (Name)o;
        int lastCmp = lastName.compareTo(n.lastName);
        return (lastCmp!=0 ? lastCmp :
```
public static void main(String[] args) {
    Name n[] = {
        new Name("Bond", "James"),
        new Name("Jack", "Blues"),
        new Name("Elwood", "Blues"),
        new Name("You", "Me")
    };
    List l = Arrays.asList(n);
    Collections.sort(l);
    System.out.println(l);
}

See also

14.24. Filling a HashTable and using a reasonable hashfunction

- **Object.hashCode contract**: s1.equals(s2) implies that s1.hashCode()==s2.hashCode() for any two sets s1 and s2.

```java
public int hashCode() {
    return 31*super.firstName.hashCode() + super.lastName.hashCode();
}
```

```java
import java.util.*;

public class Hash_1 extends Name_1 {
    static final int MAX = 20000;
    static HashMap aHashMap = new HashMap();

    public Hash_1(String firstName, String lastName) {
        super(firstName, lastName);
    }

    public static void init() {
        long milliSeconds = System.currentTimeMillis();
        for (int index = 0; index <= MAX; index++) {
            if (index % 1000 == 0)
                System.out.println(index + " / " + MAX);
            aHashMap.put(new Hash_1("A" + index, "A" + index),
                new Hash_1("A" + index, "A" + index));
        }
        System.out.println("Time for filling: " +
            (System.currentTimeMillis() - milliSeconds));
    }
}
```
public static void findIt(Hash_1 aHash_1) {
    long milliSeconds = System.currentTimeMillis();
    if (aHashMap.containsKey(aHash_1))
        System.out.print("aHashMap: containsKey takes: ");
    System.out.println(System.currentTimeMillis() - milliSeconds);
}

public static void findMax() {
    Hash_1 aHash_1 = new Hash_1("A" + MAX, "A" + MAX);
    System.out.println("Find Max = " + aHash_1);
    findIt(aHash_1);
}

public static void findMiddle() {
    Hash_1 aHash_1 = new Hash_1("A" + (MAX/2), "A" + (MAX/2));
    System.out.println("Find Middle = " + aHash_1);
    findIt(aHash_1);
}

public static void findMin() {
    Hash_1 aHash_1 = new Hash_1("A" + 0, "A" + 0);
    System.out.println("Find Min = " + aHash_1);
    findIt(aHash_1);
}

public static void main(String args[]) {
    long milliSeconds = System.currentTimeMillis();
    init();
    findMax();
    findMiddle();
    findMin();
    System.exit(0);
}

Source Code: Src/9/Hash_1.java

Time for filling: 1638
Find Max = A20000 A20000
    aHashMap: containsKey takes: 0
Find Middle = A10000 A10000
    aHashMap: containsKey takes: 0
Find Min = A0 A0
    aHashMap: containsKey takes: 0

14.25. Filling a Hash Table and not using a reasonable hash function
public int hashCode() {
    return 1;
}

import java.util.*;

public class Hash_2 extends Name_2 {
    static final int MAX = 20000;
    static HashMap aHashMap = new HashMap();

    public Hash_2(String firstName, String lastName) {
        super(firstName, lastName);
    }

    public static void init() {
        long milliSeconds = System.currentTimeMillis();
        for (int index = 0; index <= MAX; index++) {
            if (index % 3000 == 0)
                System.out.println(new Date() + ": " + index + "/" + MAX);
            aHashMap.put(new Hash_2("A" + index, "A" + index),
                        new Hash_2("A" + index, "A" + index));
        }
        System.out.println("Time for filling: " +
                           (System.currentTimeMillis() - milliSeconds));
    }

    public static void findIt(Hash_2 aHash_2) {
        long milliSeconds = System.currentTimeMillis();
        if (aHashMap.containsKey(aHash_2))
            System.out.println("aHashMap: containsKey takes: " +
                               (System.currentTimeMillis() - milliSeconds));
    }

    public static void findMax() {
        Hash_2 aHash_2 = new Hash_2("A" + MAX, "A" + MAX);
        System.out.println("Find Max = " + aHash_2);
        findIt(aHash_2);
    }

    public static void findMiddle() {
        Hash_2 aHash_2 = new Hash_2("A" + (MAX / 2), "A" + (MAX / 2));
        System.out.println("Find Middle = " + aHash_2);
        findIt(aHash_2);
    }

    public static void findMin() {
        Hash_2 aHash_2 = new Hash_2("A" + 0, "A" + 0);
        System.out.println("Find Min = " + aHash_2);
        findIt(aHash_2);
    }
}
public static void main(String args[]) {
    long milliSeconds = System.currentTimeMillis();
    init();
    findMax();
    findMiddle();
    findMin();
    System.exit(0);
}

Source Code: Src/9/Hash_2.java

% java Hash_2
Wed Sep 18 12:25:54 EDT 2002: 0/20000
Wed Sep 18 12:25:58 EDT 2002: 3000/20000
Wed Sep 18 12:26:08 EDT 2002: 6000/20000
Wed Sep 18 12:26:31 EDT 2002: 9000/20000
Wed Sep 18 12:27:11 EDT 2002: 12000/20000
Wed Sep 18 12:27:58 EDT 2002: 15000/20000
Wed Sep 18 12:29:01 EDT 2002: 18000/20000
Time for filling: 252173
Find Max = A20000 A20000
    aHashMap: containsKey takes: 1
Find Middle = A10000 A10000
    aHashMap: containsKey takes: 85
Find Min = A0 A0
    aHashMap: containsKey takes: 16

14.26. Routine Data Manipulation

The Collections class provides three algorithms for doing routine data manipulation on List objects.

- reverse: Reverses the order of the elements in a List.
- fill: Overwrites every element in a List with the specified value.
- copy: Takes two arguments, a destination List and a source List, and copies the elements of the source into the destination.
  - See java doc ...

14.27. Sorting Maps

- A Map is not sorted
- Collections.sort():
  - public static void sort(List list)
  - Convert Maps to Lists.
1 import java.util.*;
2
3 public class UseCollectionS {
4     static ArrayList aArrayList = new ArrayList();
5     static HashMap aHashMap = new HashMap();
6
7     public static void main(String args[]) {
8         for (int index = 0; index < args.length; ++index)
9             aHashMap.put(args[index], args[index] + " " + new Date());
10         System.out.println("The HashMap: " + aHashMap);
11         List l = new ArrayList(aHashMap.values());
12         Collections.sort(l);
13         System.out.println("The List: " + l);
14     }
15 }
16
Source Code: Src/9/UseCollectionS.java

javac UseCollectionS.java && java UseCollectionS a b c d

14.28. Shuffling
The shuffle algorithm does the opposite of what sort does:
• it destroys any trace of order that may have been present in a List.
• It’s useful in implementing games of chance.
• It’s useful for generating test cases.

14.29. Searching
The binarySearch algorithm searches for a specified element in a sorted List using the binary search algorithm.

1 import java.util.*;
2
3 public class Name implements Comparable {
4     protected String firstName, lastName;
5
6     public Name(String firstName, String lastName) {
7         if (firstName==null || lastName==null)
8             throw new NullPointerException();
9         this.firstName = firstName;
10         this.lastName = lastName;
11     }
12
13     public String firstName() {
14         return firstName;
15     }
16 }
17
18 Source Code: Src/9/Name.java
public String lastName()  {
    return lastName;
}
	public boolean equals(Object o) {
	    if (!(o instanceof Name))
	        return false;
	    Name n = (Name)o;
	    return n.firstName.equals(firstName) &&
	           n.lastName.equals(lastName);
	}

public String toString() {
    return firstName + " " + lastName;
}

public int compareTo(Object o) {
    Name n = (Name)o;
    int lastCmp = lastName.compareTo(n.lastName);
    return (lastCmp!=0 ? lastCmp :
               firstName.compareTo(n.firstName));
}

public static void main(String args[]) {
    Name n[] = {
        new Name("Bond", "James"),
        new Name("Jack", "Blues"),
        new Name("Elwood", "Blues"),
        new Name("You", "Me")
    };
    List l = Arrays.asList(n);
    Collections.sort(l);
    System.out.println(l);
}

Source Code: Src/9/Name.java

import java.util.*;

public class Name_1 extends Name {
    static final int MAX = 5;

    public Name_1(String firstName, String lastName) {
        super(firstName, lastName);
    }

    public int hashCode() {
        return 31*super.firstName.hashCode() + super.lastName.hashCode();
    }
}
public static void main(String args[]) {
    HashMap aHashMap = new HashMap();
    long milliSeconds = System.currentTimeMillis();

    milliSeconds = System.currentTimeMillis();
    for (int i = 1; i < MAX; i++) {
        System.out.println("1: "+i);
        for (int index = 0; index < 10000; index++)
            aHashMap.put(new Name_1("A"+index, "A"+index),
                        new Name_1("B"+index, "A"+index));
    }
    System.out.println("Time Name_1: " + (System.currentTimeMillis() - milliSeconds));
}

Source Code: Src/9/Name_1.java

import java.util.*;

public class SortTest extends Name_1 {
    static final int MAX = 20000;
    static HashMap aHashMap = new HashMap();
    static TreeMap aTreeMap = new TreeMap();
    static ArrayList aArrayList = new ArrayList();

    public SortTest(String firstName, String lastName) {
        super(firstName, lastName);
    }

    public static void init() {
        long milliSeconds = System.currentTimeMillis();
        for (int index = 0; index <= MAX; index++) {
            if (index % 1000 == 0)
                System.out.println(index + "/" + MAX);
            aTreeMap.put(new SortTest("A"+index, "A"+index),
                         new SortTest("A"+index, "A"+index));
            aHashMap.put(new SortTest("A"+index, "A"+index),
                         new SortTest("A"+index, "A"+index));
            aArrayList.add(new SortTest("A"+index, "A"+index));
        }
        System.out.println("Time for filling: " +
                           (System.currentTimeMillis() - milliSeconds));
    }
}
public static void findIt(SortTest aSortTest) {
    long milliSeconds = System.currentTimeMillis();
    if ( aArrayList.contains( aSortTest ) )
        System.out.print("aArrayList: contains takes: ");
    System.out.println( System.currentTimeMillis() - milliSeconds);
    milliSeconds = System.currentTimeMillis();
    if ( aArrayList.indexOf( aSortTest ) >= 0 )
        System.out.print("aArrayList: indexOf takes: ");
    System.out.println(System.currentTimeMillis() - milliSeconds);
    milliSeconds = System.currentTimeMillis();
    if ( aHashMap.containsKey( aSortTest ) )
        System.out.print("aHashMap: containsKey takes: ");
    System.out.println(System.currentTimeMillis() - milliSeconds);
    milliSeconds = System.currentTimeMillis();
    if ( aTreeMap.containsKey( aSortTest ) )
        System.out.print("aTreeMap: containsKey takes: ");
    System.out.println(System.currentTimeMillis() - milliSeconds);
}

class SortTest {
    String text;
    SortTest(String text) {
        this.text = text;
    }
    SortTest() {
        this.
    }
    String toString() {
        return text;
    }
}

public static void findMax() {
    SortTest aSortTest = new SortTest( "A" + MAX, "A" + MAX);
    System.out.println("Find Max = " + aSortTest);
    findIt(aSortTest);
}

public static void findMiddle() {
    SortTest aSortTest = new SortTest( "A" + (MAX/2), "A" + (MAX/2));
    System.out.println("Find Middle = " + aSortTest);
    findIt(aSortTest);
}

public static void findMin() {
    SortTest aSortTest = new SortTest( "A" + 0, "A" + 0);
    System.out.println("Find Min = " + aSortTest);
    findIt(aSortTest);
}

public static void main(String args[] ) {
    long milliSeconds = System.currentTimeMillis();
    init();
    findMax();
    findMiddle();
    findMin();
    System.exit(0);
}
Source Code: Src/9/SortTest.java

0/20000
1000/20000
2000/20000
3000/20000
4000/20000
5000/20000
6000/20000
7000/20000
8000/20000
9000/20000
10000/20000
11000/20000
12000/20000
13000/20000
14000/20000
15000/20000
16000/20000
17000/20000
18000/20000
19000/20000
20000/20000

Time for filling: 128
Find Max = A20000 A20000
  aArrayList: contains takes: 5
  aArrayList: indexOf takes: 1
  aHashMap: containsKey takes: 0
  aTreeMap: containsKey takes: 0
Find Middle = A10000 A10000
  aArrayList: contains takes: 1
  aArrayList: indexOf takes: 0
  aHashMap: containsKey takes: 0
  aTreeMap: containsKey takes: 0
Find Min = A0 A0
  aArrayList: contains takes: 0
  aArrayList: indexOf takes: 0
  aHashMap: containsKey takes: 0
  aTreeMap: containsKey takes: 0

14.30. Finding Extreme Values
The min and max algorithms return, respectively, the minimum and maximum element contained in a specified Collection.
14.31. Comparable
How about comparing based on different criterias?

• See also
• The Comparable interface consists of a single method:

```java
public interface Comparable {
    public int compareTo(Object o);
}
```

•

```
a < b    → a.compareTo(b) < 0
a == b   → a.compareTo(b) == 0
a > b    → a.compareTo(b) > 0
```

• Lists and arrays of objects that implement this interface can be sorted automatically by

```java
Collections.sort()
```

• A class’s natural ordering is said to be consistent with equals if and only if

```
(e1.compareTo((Object)e2)==0)
```

has the same boolean value as

```
e1.equals((Object)e2)
```

for every e1 and e2 of class C.

• It is strongly recommended, but not strictly required that

```
(x.compareTo(y)==0) == (x.equals(y))
```

• o = (name, age)
  — equals: name + age
  — compareTo: name

• must ensure that
  — \( \text{sgn}(x\text{.compareTo}(y)) == -\text{sgn}(y\text{.compareTo}(x)) \) for all x and y.
  — \( (x\text{.compareTo}(y)>0 \&\& y\text{.compareTo}(z)>0) \) implies \( x\text{.compareTo}(z)>0 \).
  — \( x\text{.compareTo}(y)==0 \) implies that \( \text{sgn}(x\text{.compareTo}(z)) == \text{sgn}(y\text{.compareTo}(z)) \), for all z.

14.32. Example I:

• Let’s take a look at this example:
/*  
 * "Note: this class has a natural ordering  
 * that is inconsistent with equals.  
 */

import java.util.*;

public class ComparableEx implements Comparable {
    protected String firstName;
    protected String lastName;

    public ComparableEx(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public boolean equals(Object o) {
        if (!(o instanceof ComparableEx))
            return false;
        ComparableEx n = (ComparableEx)o;
        return firstName.equals(n.firstName) && lastName.equals(n.lastName);
    }

    public int compareTo(Object o) {
        ComparableEx n = (ComparableEx)o; // cast exception
        return lastName.compareTo(lastName);
    }

    public String toString() {
        return firstName + "/" + lastName;
    }

    public static void main(String args[]) {
        ComparableEx n[] = {
            new ComparableEx("James", "Bond"),
            new ComparableEx("James", "Bond"),
            new ComparableEx("Jack", "Blues"),
            new ComparableEx("Elwood", "Blues")
        };
        List l = Arrays.asList(n);
        Collections.sort(l);
        System.out.println(l);
    }
}

Source Code: Src/9/ComparableEx.java

Es wird ein Teil der Information des kompletten Objects verwendet.

14.33. Warning
What is the problem here?
Nur ein Nachname wird eingefuegt... doppelte Nachnamen werde ignoriert...
import java.util.*;

public class ComparatorExTree {
    protected String firstName;
    protected String lastName;

    static final Comparator nameC = new Comparator() {
        public int compare(Object o1, Object o2) {
            ComparatorExTree n1 = (ComparatorExTree)o1; // cast exception
            ComparatorExTree n2 = (ComparatorExTree)o2; // cast exception
            return n1.lastName.compareTo(n2.lastName);
        }
    };

    public ComparatorExTree(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public boolean equals(Object o) {
        if (!(o instanceof ComparatorExTree))
            return false;
        ComparatorExTree n = (ComparatorExTree)o;
        return firstName.equals(n.firstName) && lastName.equals(n.lastName);
    }

    public String toString() {
        return firstName + "; " + lastName;
    }

    public int compareTo(Object o) {
        ComparableEx n = (ComparableEx)o; // cast exception
        if (firstName.compareTo(firstName) == 0)
            return lastName.compareTo(lastName);
        else
            return 0;
    }

    public static void main(String args[]) {
        ComparatorExTree n[] = {
            new ComparatorExTree("You", "Name"),
            new ComparatorExTree("Roger", "Bond"),
            new ComparatorExTree("James", "Bond"),
            new ComparatorExTree("Jack", "Blues"),
            new ComparatorExTree("Elwood", "Blues")
        };
        TreeSet l = new TreeSet(nameC);
        for (int i = 0; i < n.length; i++) {
            System.out.println(i + " " + n[i]);
        }
    }
}
54         l.add(n[i]);
55     }
56     System.out.println("the TreeSet: "+l);
57 }
58 }
59
Source Code: Src/9/ComparatorExTree.java

java ComparatorExTree
0 You; Name
1 Roger; Bond
2 James; Bond
3 Jack; Blues
4 Elwood; Blues
the TreeSet: [Jack; Blues, Roger; Bond, You; Name]

14.34. Comparator

(From java doc)
The relation that defines the imposed ordering that a given comparator c imposes on a given set of objects S is:
For all x, y \in S:
\{(x, y) such that c.compare(x, y) \leq 0\}.
The quotient for this total order is:
\{(x, y) such that c.compare(x, y) = 0\}.

14.35. Example

1        import java.util.*;
2
3        class HpComparator implements Comparator {
4             public int compare(Object o1, Object o2) {
5                 String s1 = (String)o1;
6                 String s2 = (String)o2;
7                 return s1.compareTo(s2);
8             }
9         }
10        public boolean equals(Object o) {
11             return true;
12         }
13     }
14     public class HpC_C_ex {
15             public static void sort(List aList) {
16                 Object anArray[] = aList.toArray();
17                 for (int index=0; index<anArray.length - 1; index++) {
18                     for (int walker=0; walker<anArray.length - index - 1; walker++)
19                         Comparable left = (Comparable) anArray[walker];
20                         Comparable right = (Comparable) anArray[walker+1];
21                         if (left.compareTo(right) > 0) {
22                             
23                         
24                     }
25                 }
26             }
27     }

Object tmp = anArray[walker];
anArray[walker] = anArray[walker + 1];
anArray[walker+1] = tmp;
}

ListIterator anIterator = aList.listIterator();
for (int j=0; j<anArray.length; j++) {
anIterator.next();
anIterator.set(anArray[j]);
}

public static void sort(List aList, Comparator aComparator) {
    Object anArray[] = aList.toArray();

    for (int index=0; index<anArray.length - 1; index++) {
        for (int walker=0; walker<anArray.length - index - 1; walker++)
            Object left = anArray[walker];
            Object right = anArray[walker+1];
            if (aComparator.compare(left, right) > 0) {
                Object tmp = anArray[walker];
                anArray[walker] = anArray[walker + 1];
                anArray[walker+1] = tmp;
            }
    }

    ListIterator anIterator = aList.listIterator();
    for (int j=0; j<anArray.length; j++) {
        anIterator.next();
anIterator.set(anArray[j]);
    }
}

public static void main(String args[]) {
    args = new String[4];

    List l = Arrays.asList(args);
    HpC_C_ex.sort(l);
    System.out.println("1. " + l);

    l = Arrays.asList(args);
    HpC_C_ex.sort(l, new HpComparator());
    System.out.println("2. " + l);
}

Source Code: Src/9/HpC_C_ex.java
14.36. Collections.sort()

```java
import java.util.*;

public class HpComparator {
    public static void sort(List list) {
        Object anArray[] = list.toArray();
        // Arrays.sort(a, c); ////////// this is the trick
        for (int index=0; index<anArray.length - 1; index++) {
            for (int walker=0; walker<anArray.length - index - 1; walker++) {
                Object left = anArray[walker];
                Object right = anArray[walker+1];
                if (left.compareTo(right) > 0) {
                    Object tmp = anArray[walker];
                    anArray[walker] = anArray[walker+1];
                    anArray[walker+1] = tmp;
                }
            }
        }
        ListIterator i = list.listIterator();
        for (int j=0; j<anArray.length; j++) {
            i.next();
            i.set(anArray[j]);
        }
    }
    public static void sort(List list, Comparator c) {
        Object anArray[] = list.toArray();
        // Arrays.sort(a, c); ////////// this is the trick
        for (int index=0; index<anArray.length - 1; index++) {
            for (int walker=0; walker<anArray.length - index - 1; walker++) {
                Object left = anArray[walker];
                Object right = anArray[walker+1];
                if (c.compare(left, right) > 0) {
                    Object tmp = anArray[walker];
                    anArray[walker] = anArray[walker+1];
                    anArray[walker+1] = tmp;
                }
            }
        }
    }
}
```

Source Code: Src/9/HpComparator_1.java
Function pointer in C/C++

14.37. Comparator in separate Classes

The use:

```java
import java.util.*;

public class ComparatorExTreeClass {
    static HpbComparator theNth = new HpbComparator();

    protected static int soManyS;
    protected String name;
    protected int waitingListN;

    public ComparatorExTreeClass(String name) {
        if (name==null)
            throw new NullPointerException();
        this.name = name;
        this.waitingListN = soManyS ++;
    }

    public ComparatorExTreeClass(String name, int waitingListN) {
        this(name);
        this.waitingListN = waitingListN;
    }

    public String getName() {
        return name;
    }

    public String toString() {
        return name + " - " + waitingListN;
    }

    public static void main(String args[]) {
        WaitingList n[] = {
```
new WaitingList("Bond"),
new WaitingList("Jack"),
new WaitingList("Elwood"),
new WaitingList("You", -1),
new WaitingList("Me", -1)
);
TreeSet l = new TreeSet(theNth);
for ( int i = 0; i < n.length; i ++ ) {
    System.out.println(i + " " + n[i]);
    l.add(n[i]);
}
System.out.println("the TreeSet: " + l);
}
}

Source Code: Src/9/ComparatorExTreeClass.java

The comparator:

import java.util.*;

public class HpbComparator implements Comparator {

    public int compare(Object o1, Object o2) {
        if ( ( o1 instanceof WaitingList ) &&
            ( o2 instanceof WaitingList ) ) {
            WaitingList n1 = (WaitingList)o1;
            WaitingList n2 = (WaitingList)o2;
            int nameCompareV = n1.name.compareTo(n2.name);
            return ( nameCompareV == 0 ?
                n1.waitingListN - n2.waitingListN :
                nameCompareV );
        } else
            return -1;
    }
}

Source Code: Src/9/HpbComparator.java
Questions:
   — Which classes do you need?
   — Comparator, do you need them?
   — which kind of collections do you need?

14.38. Exercise II
• What do you have to be aware of, if you write a compareTo method?
• What do you have to be aware of, if you write a compare method?
15. Reflection API

Most of the stuff 'stolen' from

The represents, or reflects, the classes, interfaces, and objects in the current Java Virtual Machine.

Typical use:

- debuggers,
- class browsers,
- GUI builders.

With the reflection API you can:

- Determine the class of an object.
- Get information about a class’s modifiers, fields, methods, constructors, and superclasses.
- Find out what constants and method declarations belong to an interface.
- Create an instance of a class whose name is not known until runtime.
- Get and set the value of an object’s field, even if the field name is unknown to your program until runtime.
- Invoke a method on an object, even if the method is not known until runtime.
- Create a new array, whose size and component type are not known until runtime, and then modify the array’s component.
### 15.1. Intro example

```java
import java.lang.reflect.*;

class Intro {
    static void printName(Object o) {
        Class c = o.getClass();
        String s = c.getName();
        System.out.println(s);
    }

    public static void main(String[] args) {
        String aS = new String();
        printName(aS);
    }
}
```

**Result:**

```
% javac Intro.java
% java Intro
java.lang.String
```
15.2. Discovering Class Modifiers

See

```java
import java.lang.reflect.*;

public final class MyModifier {

    void printName(Object o) {
        Class c = o.getClass();
        String s = c.getName();
        System.out.println(s);
    }

    public void printModifiers(Object o) {
        Class c = o.getClass();
        int m = c.getModifiers();
        if (Modifier.isPrivate(m))
            System.out.println("private");
        if (Modifier.isPublic(m))
            System.out.println("public");
        if (Modifier.isAbstract(m))
            System.out.println("abstract");
        if (Modifier.isFinal(m))
            System.out.println("final");
    }

    public static void main(String[] args) {
        MyModifier aM = new MyModifier();
        aM.printName(aM);
        aM.printModifiers(aM);
    }
}
```

Result:

```
% java MyModifier
MyModifier
public
final
```
15.3. Identifying Class Fields

Application such as a class browser, might want to find out what fields belong to a particular class. This can be identified by invokesing the getFields method on a Class object. The getFields method returns an array of Field objects containing one object per accessible public field.

A public field is accessible if it is a member of either:

- this class
- a superclass of this class
- an interface implemented by this class
- an interface extended from an interface implemented by this class

```java
import java.lang.reflect.*;

class What {
    public int publicVar;;
    private int privateVar;;
    static int staticVar;;

    static void printFieldNames(Object o) {
        Class c = o.getClass();
        Field[] publicFields = c.getFields();
        for (int i = 0; i < publicFields.length; i++) {
            String fieldName = publicFields[i].getName();
            Class typeClass = publicFields[i].getType();
            String fieldType = typeClass.getName();
            System.out.println("Name: " + fieldName + ", Type: " + fieldType);
        }
    }

    public static void main(String[] args) {
        String aS = new String();
        Thread aT = new Thread();
        What aW = new What();
        System.out.println("String: ");
        printFieldNames(aS);
        System.out.println("Thread: ");
        printFieldNames(aT);
        System.out.println("What: ");
        printFieldNames(aW);
    }
}
```

Result:

% java What
String:
    Name: serialVersionUID, Type: long
    Name: CASE_INSENSITIVE_ORDER, Type: java.util.Comparator
Thread:
Name: MIN_PRIORITY, Type: int
Name: NORM_PRIORITY, Type: int
Name: MAX_PRIORITY, Type: int

What:
Name: publicVar, Type: int
15.4. Getting Values

A development tool such as a debugger, must be able to obtain field values. This is a three-step process:

1. Create a Class object.
2. Create a Field object by invoking getField on the Class object.
3. Invoke one of the get methods on the Field object.

```java
import java.lang.reflect.*;

class Get {
    public int publicVar = 42;
    private int privateVar;
    static int staticVar;

    static void getValue(Object o) {
        Class c = o.getClass();
        Integer value;
        try {
            Field publicVarField = c.getField("publicVar");
            value = (Integer) publicVarField.get(o);
            System.out.println("value: "+ value);
        } catch (NoSuchFieldException e) {
            System.out.println(e);
        } catch (SecurityException e) {
            System.out.println(e);
        } catch (IllegalAccessException e) {
            System.out.println(e);
        }
    }

    public static void main(String[] args) {
        Get aG = new Get();
        System.out.println("Get: ");
        getValue(aG);
    }
}

% java Get
Get:
value: 42
```
15.5. Setting Values

Some debuggers allow users to change field values during a debugging session. A tool that has this capability, must call one of the Field class’s set methods.

1. To modify the value of a field, perform the following steps: Create a Class object. For more information, see the section Retrieving Class Objects.
2. Create a Field object by invoking getField on the Class object.
3. Class Fields shows you how. Invoke the appropriate set method on the Field object.

The Field class provides several set methods. Specialized methods, such as setBoolean and setInt, are for modifying primitive types. If the field you want to change is an object invoke the set method. It is possible to set to modify a primitive type, but the appropriate wrapper object for the value parameter must be used.

```java
1 2 import java.lang.reflect.*;
3
4 class Set {
5   public int publicVar = 42;
6   private int privateVar;
7   static int staticVar;
8
9   static void setValue(Object o) {
10      Class c = o.getClass();
11      Integer value;
12      try {
13         Field publicVarField = c.getField("publicVar");
14         publicVarField.set(o, new Integer(24));
15      } catch (NoSuchFieldException e) {
16         System.out.println(e);
17      } catch (SecurityException e) {
18         System.out.println(e);
19      } catch (IllegalAccessException e) {
20         System.out.println(e);
21      }
22   }
23
24   public static void main(String[] args) {
25      Set aS = new Set();
26
27      System.out.println("before: aS.publicVar = " + aS.publicVar);
28      setValue(aS);
29      System.out.println("after: aS.publicVar = " + aS.publicVar);
30   }
31
% java Set
before: aS.publicVar = 42
after: aS.publicVar = 24
```
15.6. Obtaining Method Information

To find out what public methods belong to a class, invoke the method named getMethods. The array returned by getMethods contains Method objects. This can be used to uncover a method’s name, return type, parameter types, set of modifiers, and set of throwable exceptions. All of this information would be useful to write a class browser or a debugger. A method can be called with Method.invoke.

1. It retrieves an array of Method objects from the Class object by calling getMethods.
2. For every element in the Method array, the program:
   a. retrieves the method name by calling getName
   b. gets the return type by invoking getReturnType
   c. creates an array of Class objects by invoking getParameterTypes
3. The array of Class objects created in the preceding step represents the parameters of the method. To retrieve the class name for every one of these parameters, the program invokes getName against each Class object in the array.

```java
import java.lang.reflect.*;

class Show {
    public int publicVar = 42;
    private int privateVar;
    static int staticVar;

    public static int HPclassM(String aString) {
        return 1;
    }

    public int HPObjM(int i, Boolean aBoolean) {
        return 1;
    }
}

void showMethods(Object o) {
    Class c = o.getClass();
    Method[] theMethods = c.getMethods();
    for (int i = 0; i < theMethods.length; i++) {
        String methodString = theMethods[i].getName();
        System.out.println("Name: " + methodString);
        String returnString =
            theMethods[i].getReturnType().getName();
        System.out.println(" Return Type: " + returnString);
        Class[] parameterTypes = theMethods[i].getParameterTypes();
        System.out.println(" Parameter Types:");
        for (int k = 0; k < parameterTypes.length; k++) {
            String parameterString = parameterTypes[k].getName();
            System.out.println(" " + parameterString);
        }
    }
}

public static void main(String[] args) {
    Show aS = new Show();
    System.out.println("Show: ");
}
```
38 aS.showMethods(aS);
39
40 }
41 }

% java Show
Show:
Name: HPclassM
  Return Type: int
  Parameter Types: java.lang.String
Name: main
  Return Type: void
  Parameter Types: [Ljava.lang.String;
  ...
Name: wait
  Return Type: void
  Parameter Types: long int
Name: HPobjectM
  Return Type: int
  Parameter Types: int java.lang.Boolean
15.7. Invoking Methods

A debugger should halted a user to select and then invoke methods during a debugging session. Since it is not known at compile time which methods the user will invoke, the method name cannot be hardcoded in the source code.

1. Create a Class object that corresponds to the object whose method you want to invoke. See the section Retrieving Class Objects for more information.

2. Create a Method object by invoking getMethod on the Class object. The getMethod method has two arguments: a String containing the method name, and an array of Class objects. Each element in the array corresponds to a parameter of the method you want to invoke. For more information on retrieving Method objects, see the section Obtaining Method Information.

3. Invoke the method by calling invoke. The invoke method has two arguments: an array of argument values to be passed to the invoked method, and an object whose class declares or inherits the method.

```java
import java.lang.reflect.*;

class Invoke {
    public int publicVar = 42;
    private int privateVar;
    static int staticVar;

    public String objectM(Boolean aBoolean, Integer i) {
        System.out.println("objectM: i = " + i);
        System.out.println("objectM: aBoolean = " + aBoolean);
        return "Alles Verloren, alles Geboren ...";
    }

    void invokeMethod(Object o) {
        Class c = Invoke.class;
        Class[] parameterTypes = new Class[] {Boolean.class, Integer.class};
        Method objectM;
        Object[] arguments = new Object[] {new Boolean(true), new Integer(2)};
        try {
            objectM = c.getMethod("objectM", parameterTypes);
            System.out.println((String) objectM.invoke((Invoke)o, arguments));
        } catch (NoSuchMethodException e) {
            System.out.println(e);
        } catch (IllegalAccessException e) {
            System.out.println(e);
        } catch (InvocationTargetException e) {
            System.out.println(e);
        }
    }

    public static void main(String[] args) {
```
invoke aS = new Invoke();
System.out.println("Invoke: ");
aS.invokeMethod(aS);

% java Invoke
Invoke:
objectM: i = 2
objectM: aBoolean = true
Alles Verloren, alles Geboren ...

16. Swing

Note: Most of the stuff stolen from or from the jdk documentation. Most programs are modified or written by me.

This section explains the various elements of the graphical user interface, i.e., most of the classes of the Swing using a set of very primitive applications. The classes are part of the package.

Note: "Swing" was the codename of the project that developed the new components. Although it’s an unofficial name, it’s frequently used to refer to the new components and related API. It’s immortalized in the package names for the Swing API, which begin with javax.swing.
16.1. JFC

JFC stands for Java Foundation Classes, which encompass a group of features to help people build graphical user interfaces (GUIs). The JFC was first announced at the 1997 JavaOne developer conference and is defined as containing the following features:

The Swing Components
   Include everything from buttons to split panes to tables. You can see mugshots of all the components in A Visual Index to the Swing Components.

Pluggable Look & Feel Support
   Gives any program that uses Swing components a choice of look and feel. For example, the same program can use either the Java Look & Feel or the Windows Look & Feel. We expect many more look-and-feel packages -- including some that use sound instead of a visual "look" -- to become available from various sources.

Accessibility API
   Enables assistive technologies such as screen readers and Braille displays to get information from the user interface.

Java 2D API (JDK 1.2 only)
   Enables developers to easily incorporate high-quality 2D graphics, text, and images in applications and in applets.

Drag and Drop Support (JDK 1.2 only)
   Provides the ability to drag and drop between a Java application and a native application.

The first three JFC features were implemented without any native code, relying only on the API defined in JDK 1.1. As a result, they could and did become available as an extension to JDK 1.1. This extension was released as JFC 1.1, which is sometimes called "the Swing release." The API in JFC 1.1 is often called the Swing API.

Please see also
16.2. SwingSet 3

Download it from

% cd /usr/local/jdk/demo/jfc/SwingSet3 && java -jar SwingSet2.jar
% appletviewer SwingSet2.html

- SwingSet.html:

```html
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
<head>
  <title>SwingSet demo</title>
</head>

<body>
  <h1>SwingSet demo</h1>
  <applet code=SwingSet2Applet codebase="." archive="SwingSet3.jar" width=695 height=525>
  </applet>
</body>
</html>
```

16.3. Top-Level Containers

The components at the top of any Swing containment hierarchy.

It exists mainly to provide a place for other Swing components to paint themselves. The commonly used top-level containers are frames (JFrame), dialogs (JDialog), and applets (JApplet).

The following statement can be found here: but it does not work in MacOS 10/Sun OS 5.9.

To view the containment hierarchy for any frame or dialog, click its border to select it, and then press Control-Shift-F1. A list of the containment hierarchy will be written to the standard output stream.

A visual index to the swing components can be found at:

16.4. General-Purpose Containers

Intermediate containers that can be used under many different circumstances.
16.5. Special-Purpose Containers
Intermediate containers that play specific roles in the UI.
16.6. Basic Controls
Atomic components that exist primarily to get input from the user; they generally also display some simple state.
16.7. Uneditable Information Displays

Atomic components that exist solely to give the user information.
16.8. Editable Displays of Formatted Information
Atomic components that display highly formatted information that (if you choose) can be edited by the user.
16.9. Swing versus AWT

The AWT components are those provided by the JDK 1.0 and 1.1 platforms. Although JDK 1.3 still supports the AWT components, SUN strongly encourages to use Swing components instead. You can identify Swing components because their names start with J.

The AWT button class, for example, is named Button, while the Swing button class is named JButton.

Additionally, the AWT components are in the java/awt package, while the Swing components are in the javax.swing package.

The biggest difference between the AWT components and Swing components is that the Swing components are implemented with absolutely no native code. Since Swing components aren’t restricted to the least common denominator — the features that are present on every platform — they can have more functionality than AWT components. Because the Swing components have no native code, they can be shipped as an add-on to JDK 1.1, in addition to being part of JDK 1.3.
Even the simplest Swing components have capabilities far beyond what the AWT components offer:

- Swing buttons and labels can display images instead of, or in addition to, text.
- You can easily add or change the borders drawn around most Swing components. For example, it’s easy to put a box around the outside of a container or label.
- You can easily change the behavior or appearance of a Swing component by either invoking methods on it or creating a subclass of it.
- Swing components don’t have to be rectangular. Buttons, for example, can be round.
- Assistive technologies such as screen readers can easily get information from Swing components. For example, a tool can easily get the text that’s displayed on a button or label.
Swing lets you specify which look and feel your program’s GUI uses. By contrast, AWT components always have the look and feel of the native platform.

CDE:
Java:
Another interesting feature is that Swing components with state use models to keep the state. A JSlider, for instance, uses a BoundedRangeModel object to hold its current value and range of legal values. Models are set up automatically, so you don’t have to deal with them unless you want to take advantage of the power they can give you.

Swing components aren’t thread safe. If you modify a visible Swing component -- invoking its setText method, for example -- from anywhere but an event handler, then you need to take special steps to make the modification execute on the event-dispatching thread. This isn’t an issue for many Swing programs, since component-modifying code is typically in event handlers.

The containment hierarchy for any window or applet that contains Swing components must have a Swing top-level container at the root of the hierarchy. For example, a main window should be implemented as a JFrame instance rather than as a Frame instance.

You don’t add components directly to a top-level container such as a JFrame. Instead, you add components to a container (called the content pane) that is itself contained by the JFrame.
16.10. The First Swing Program

```java
import javax.swing.*; //This is the final package name.
import java.awt.*;
import java.awt.event.*;

public class SwingApplication {
    private static String labelPrefix = "So many clicks: ";
    private int numClicks = 0;

    public Component createComponents() {
        final JLabel label = new JLabel(labelPrefix + "0 ");
        JButton button = new JButton("Swing button!");
        button.addActionListener(
            new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    numClicks++;
                    label.setText(labelPrefix + numClicks);
                    System.out.println("getActionCommand: " + e.getActionCommand());
                    System.out.println("getModifiers: " + e.getModifiers());
                    System.out.println("paramString: " + e.paramString());
                }
            }
        );
        label.setLabelFor(button);
        // An easy way to put space between a top-level container
        // and its contents is to put the contents in a JPanel
        // that has an "empty" border.
        JPanel pane = new JPanel();
        pane.setBorder(BorderFactory.createEmptyBorder(30, //top
                                                      30, //left
                                                      10, //bottom
                                                      30)); //right
        pane.setLayout(new FlowLayout());
        pane.add(button);
        pane.add( new JFileChooser()) ;
        pane.add(new JCheckBox("not me", true) );
        pane.add(label);
        return pane;
    }
}
```
public static void main(String[] args) {
    String lookAndFeel;
    lookAndFeel=UIManager.getCrossPlatformLookAndFeelClassName();
    if ( args.length == 1 )
    {
        if ( args[0].equals("motif") )
            lookAndFeel =
                "com.sun.java.swing.plaf.motif.MotifLookAndFeel";
        if ( args[0].equals("metal") )
            lookAndFeel =
                "javax.swing.plaf.metal.MetalLookAndFeel";
        else if ( args[0].equals("system") )
            lookAndFeel= UIManager.getSystemLookAndFeelClassName() ;
        try {
            UIManager.setLookAndFeel( lookAndFeel);
        } catch (Exception e) { }
        UIManager.setLookAndFeel( lookAndFeel);
    }
    try {
        UIManager.setLookAndFeel( lookAndFeel);
    } catch (Exception e) { }
    //Create the top-level container and add contents to it.
    JFrame frame = new JFrame("SwingApplication");
    SwingApplication app = new SwingApplication();
    Component contents = app.createComponent();
    frame.getContentPane().add(contents);
    //Finish setting up the frame, and show it.
    frame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
    frame.pack();
    frame.setVisible(true);
}

Source Code: Src/12/SwingApplication.java

Output:
getActionCommand: Swing button!
getModifiers: 0
paramString: ACTION_PERFORMED,cmd=Swing button!
getActionCommand: Swing button!
getModifiers: 0
paramString: ACTION_PERFORMED,cmd=Swing button!
getActionCommand: Swing button!
getModifiers: 0
...
Motif:
16.11. Layout Management

Layout management is the process of determining the size and position of components. By default, each container has a layout manager -- an object that performs layout management for the components within the container. Components can provide size and alignment hints to layout managers, but layout managers have the final say on the size and position of those components.

The Java platform supplies five commonly used layout managers: BorderLayout, BoxLayout, FlowLayout, GridBagLayout, and GridLayout. These layout managers are designed for displaying multiple components at once. A sixth provided class, CardLayout, is a special-purpose layout manager used in combination with other layout managers.

Whenever you use the add method to put a component in a container, you must take the container’s layout manager into account. Some layout managers, such as BorderLayout, require you to specify the component’s relative position in the container, using an additional argument with the add method. Occasionally, a layout manager such as GridBagLayout requires elaborate setup procedures. Many layout managers, however, simply place components based on the order they were added to their container.

16.12. Layout Manager

See also

16.13. GridLayout Manager

The GridLayout class is a layout manager that lays out a container’s components in a rectangular grid.

The container is divided into equal-sized rectangles, and one component is placed in each rectangle.


GridBagLayout is the most sophisticated, flexible layout manager the Java platform provides. It aligns components by placing them within a grid of cells, allowing some components to span more than one cell. The rows in the grid aren’t necessarily all the same height; similarly, grid columns can have different widths.

16.15. FlowLayout Manager

A flow layout arranges components in a left-to-right flow, much like lines of text in a paragraph. Flow layouts are typically used to arrange buttons in a panel. It will arrange buttons left to right until no more buttons fit on the same line. Each line is centered.

16.16. Viewport Manager

The default layout manager for JViewport. JViewportLayout defines a policy for layout that should be useful for most applications. The viewport makes its view the same size as the viewport, however it will not make the view smaller than its minimum size. As the viewport grows the view is kept bottom justified until the entire view is visible, subsequently the view is kept top justified.
16.17. ScrollPaneLayout Manager

The layout manager used by JScrollPane. JScrollPaneLayout is responsible for nine components: a viewport, two scrollbars, a row header, a column header, and four "corner" components.
import javax.swing.*;  //This is the final package name.
import java.awt.*;
import java.awt.event.*;

public class Calculator {
    private JLabel label = null;
    private int numClicks = 0;

    public Component createLabel() {
        final JLabel label = new JLabel("0");
        label.setText("0");
        label.setHorizontalAlignment(JLabel.RIGHT);
        this.label = label;
        return label;
    }

    public Component createButtons() {
        JPanel pane = new JPanel();
        pane.setBorder(BorderFactory.createLoweredBevelBorder());
        pane.setLayout(new GridLayout(5, 2));
        for (int index = 0; index < 10; index++) {
            JButton button = new JButton(new Integer(index).toString());
            button.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    label.setText(e.getActionCommand());
                }
            });
            pane.add(button);
        }
        return pane;
    }

    public Component createOps() {
        JPanel pane = new JPanel();
        pane.setBorder(BorderFactory.createLoweredBevelBorder());
        pane.setLayout(new GridLayout(2, 2));
        pane.add(new JButton("+"));
        pane.add(new JButton("-"));
        pane.add(new JButton("*"));
        pane.add(new JButton("/"));
        return pane;
    }

    public Component createComponents() {
        JPanel pane = new JPanel();
        pane.setBorder(BorderFactory.createLoweredBevelBorder());
        pane.setLayout(new GridLayout(2, 2));
        pane.add(new JButton("+"));  
        pane.add(new JButton("-"));  
        pane.add(new JButton("*"));  
        pane.add(new JButton("/"));  
        return pane;
    }
}
pane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
pane.add(createLabel());
pane.add(createButtons());
pane.add(createOps());

return pane;
}

public static void main(String[] args) {
    String lookAndFeel;
    lookAndFeel=UIManager.getCrossPlatformLookAndFeelClassName();
    if (args.length == 1) {
        if (args[0].equals("motif") )
            lookAndFeel =
            "com.sun.java.swing.plaf.motif.MotifLookAndFeel";
        if (args[0].equals("metal") )
            lookAndFeel =
            "javax.swing.plaf.metal.MetalLookAndFeel";
        else if (args[0].equals("system") )
            lookAndFeel= UIManager.getSystemLookAndFeelClassName() ;
    }
    try {
        UIManager.setLookAndFeel(lookAndFeel);
    } catch (Exception e) { }

    //Create the top-level container and add contents to it.
    JFrame frame = new JFrame("Calculator");
    Calculator app = new Calculator();
    Component contents = app.createComponent();
    frame.getContentPane().add(contents);

    //Finish setting up the frame, and show it.
    frame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
    frame.pack();
    frame.setVisible(true);
}

Source Code: Src/12/Calculator.java

16.19. GridBagLayout

Essentially, GridBagLayout places components in rectangles (cells) in a grid, and then uses
the components’ preferred sizes to determine how big the cells should be.
source code:

```java
import javax.swing.*; //This is the final package name.
import java.awt.*;
import java.awt.event.*;

public class GB {

    public Component createComponents() {
        JButton button;
        JPanel contentPane = new JPanel();
        GridBagLayout gridbag = new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();
        contentPane.setLayout(gridbag);
        c.fill = GridBagConstraints.HORIZONTAL;

        button = new JButton("Button 1");
        c.weightx = 0.5;
        c.gridx = 0;
        c.gridy = 0;
        gridbag.setConstraints(button, c);
        contentPane.add(button);

        button = new JButton("2");
        c.weightx = 4;
        c.gridx = 1;
        c.gridy = 0;
        gridbag.setConstraints(button, c);
        contentPane.add(button);

        button = new JButton("Button 3");
        c.weightx = 8;
        c.gridx = 2;
        c.gridy = 0;
        gridbag.setConstraints(button, c);
        contentPane.add(button);

        button = new JButton("Long-Named Button 4");
        c.ipady = 40; //make this component tall
        c.weightx = 0.0;
        c.insets = new Insets(20,0,0,0); //top padding
        c.gridwidth = 3;
        c.gridx = 0;
        c.gridy = 1;
        gridbag.setConstraints(button, c);
        contentPane.add(button);

        button = new JButton("Button 5");
        c.ipady = 0; //reset to default
        c.weighty = 1.0; //request any extra vertical space
        c.anchor = GridBagConstraints.SOUTH; //bottom of space
        c.insets = new Insets(10,0,0,0); //top padding
    }
```
When you enlarge the window the program brings up, the columns grow proportionately. This is because each component in the first row, where each component is one column wide, has weightx = 1.0.

The actual value of these components’ weightx is unimportant. What matters is that all the components, and consequently, all the columns, have an equal weight that is greater than 0. If no component managed by the GridBagLayout had weightx set, then when the components’ container was made wider, the components would stay clumped together in the center of the container.
16.20. A Second Try
import javax.swing.*; //This is the final package name.
import java.awt.*;
import java.awt.event.*;

public class Lotto {

    public void createButtons( GridBagLayout gridbag,
                               JPanel contentPane,
                               int gridy) {
        int number = 1;

        GridBagConstraints c = new GridBagConstraints();
        c.gridwidth = 1;
        c.weightx = 1.0;
        c.insets = new Insets(5,3,3,5);
        c.fill = GridBagConstraints.BOTH;
        JButton button;
        JButton o;

        for (int col = 1; col <= 7; col++) {
            for (int row = 1; row <= 7; row++) {
                button = new JButton(
                        new Integer(number++).toString());
                button.setBackground(Color.yellow);
                button.setMinimumSize(button.getSize());
                button.addActionListener(
                        new ActionListener() {  
                            public void actionPerformed(ActionEvent e) {  
                                o = (JButton)e.getSource(); // will this compile
                                System.out.println("getText = " +  
                                        o.getText());  
                                if (o.getBackground() == Color.red)  
                                    o.setBackground(Color.yellow);  
                                else  
                                    o.setBackground(Color.red);
                            }
                        });
                c.gridx = row;
                c.gridy = col + gridy;
                gridbag.setConstraints(button, c);
                contentPane.add(button);
            }
        }
    }

    public void createSeparator( GridBagLayout gridbag,
                               JPanel contentPane,
                               int gridy) {
        GridBagConstraints c = new GridBagConstraints();
    }

    public void createButtons( GridBagLayout gridbag,
                               JPanel contentPane,
                               int gridy) {
    }
}
JSeparator sep = new JSeparator(SwingConstants.HORIZONTAL);
c.fill = GridBagConstraints.BOTH;
c.gridx = 0;
c.gridy = gridy;
c.gridwidth = 8;
c.insets = new Insets(5,3,3,5);
gridbag.setConstraints(sep, c);
contentPane.add(sep);
}

public Component createComponents() {
    JSeparator sep;
    JButton button;
    JLabel label;
    JPanel contentPane = new JPanel();
    GridBagLayout gridbag = new GridBagLayout();
    GridBagConstraints c = new GridBagConstraints();
    contentPane.setLayout(gridbag);
    label = new JLabel("0");
c.weightx = 0.5;
c.gridx = 0;
c.gridy = 0;
c.gridwidth = 8;
c.ipady = 10;
gridbag.setConstraints(label, c);
    contentPane.add(label);
    createSeparator(gridbag, contentPane, 1);
    createButtons(gridbag, contentPane, 3);
    createSeparator(gridbag, contentPane, 11);
    JLabel label_1 = new JLabel("0 correct");
c.weightx = 0.5;
c.gridx = 0;
c.gridy = 12;
c.gridwidth = 4;
c.ipady = 10;
gridbag.setConstraints(label_1, c);
contentPane.add(label_1);
    JTextField tField = new JTextField("0", 5);
tField.setHorizontalAlignment(JTextField.RIGHT);
tField.setEditable(false);
c.weightx = 0.5;
c.gridx = 4;
c.gridy = 12;
c.gridwidth = 4;
c.ipady = 10;
gridbag.setConstraints(tField, c);
contentPane.add(tField);
    createSeparator(gridbag, contentPane, 13);
return contentPane;

public static void main(String[] args) {
    String lookAndFeel = UIManager.getCrossPlatformLookAndFeelClassName();
    try {
        UIManager.setLookAndFeel(lookAndFeel);
    } catch (Exception e) { }
    JFrame frame = new JFrame("Lotto");
    Lotto app = new Lotto();
    Component contents = app.createComponent();
    frame.getContentPane().add(contents);
    //Finish setting up the frame, and show it.
    frame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
    frame.pack();
    frame.setVisible(true);
}

Source Code: Src/12/Lotto.java


- Divide the area in logical pieces
- Do the objects form a regular/grid pattern?
- What should happen if there is a change in size?

16.22. How to Choose a Layout Manager

Stolen from

- You need to display a component in as much space as it can get.
  Consider using BorderLayout or GridBagLayout. If you use BorderLayout, you’ll need to put the space-hungry component in the center. With GridBagLayout, you’ll need to set the constraints for the component so that fill=GridBagConstraints.BOTH. Another possibility is to use BoxLayout, making the space-hungry component specify very large preferred and maximum sizes.

- You need to display a few components in a compact row at their natural size.
  Consider using a JPanel to group the components and using either the JPanel’s default FlowLayout manager or the BoxLayout manager.
• You need to display a few components of the same size in rows and columns.

GridLayout is perfect for this.

• You need to display a few components in a row or column, possibly with varying amounts of space between them, custom alignment, or custom component sizes.

BoxLayout is perfect for this.

• You have a complex layout with many components. Consider either using GridBagLayout or grouping the components into one or more JPanels to simplify layout. Each JPanel might use a different layout manager.

16.23. Creating a Custom Layout Manager

Instead of using one of the Java platform’s layout managers, you can write your own. Layout managers must implement the LayoutManager interface, which specifies the five methods every layout manager must define. Optionally, layout managers can implement LayoutManager2, which is a subinterface of LayoutManager.
17. Event Handling

Every time the user types a character or presses a mouse button, an event occurs. Any object can be notified of the event. All it has to do is implement the appropriate interface and be registered as an event listener on the appropriate event source. Swing components can generate many kinds of events. Here are a few examples:

<table>
<thead>
<tr>
<th>Act that results in the event</th>
<th>Listener type</th>
</tr>
</thead>
<tbody>
<tr>
<td>User clicks a button, presses Return while typing in a text field, or chooses a menu item</td>
<td>ActionListener</td>
</tr>
<tr>
<td>User closes a frame (main window)</td>
<td>WindowListener</td>
</tr>
<tr>
<td>User presses a mouse button while the cursor is over a component</td>
<td>MouseListener</td>
</tr>
<tr>
<td>User moves the mouse over a component</td>
<td>MouseMotionListener</td>
</tr>
<tr>
<td>Component becomes visible</td>
<td>ComponentListener</td>
</tr>
<tr>
<td>Component gets the keyboard focus</td>
<td>FocusListener</td>
</tr>
<tr>
<td>Table or list selection changes</td>
<td>ListSelectionListener</td>
</tr>
</tbody>
</table>

Each event is represented by an object that gives information about the event and identifies the event source. Event sources are typically components, but other kinds of objects can also be event sources. As the following figure shows, each event source can have multiple listeners registered on it. Conversely, a single listener can register with multiple event sources.
17.1. Messaging vs. listening

(c) Axel T. Schreiner

When sending a message, i.e., calling a method for a particular receiver, arbitrary method signatures can be used and arbitrary information can be passed and returned. If necessary, java.lang.reflect could be used to delay most decisions until runtime.

When acting as a listener, the receiving method signature and associated information is fixed and nothing can be returned. However, an arbitrary number of receivers can choose to receive the same information.

Which of the techniques to use and where to implement the listener, if any, is a significant design choice, especially, because the listener can additionally be nested, affording further flexibility.

If the listener is nested on the event sender (view) side, status information can be sent, but the augmented view is probably not very general.

If the listener is nested on the event receiver (control?) side, sender decoding is implicit and significant interaction with the receiver state is possible. Additionally, different kinds of events can be mapped to the same receiver reaction.
17.2. How to Implement an Event Handler

Every event handler requires three bits of code:

1. In the declaration for the event handler class, code that specifies that the class either implements a listener interface or extends a class that implements a listener interface. For example:
   
   ```java
   public class MyClass implements ActionListener {
   ```

2. Code that registers an instance of the event handler class as a listener upon one or more components. For example:
   
   ```java
   someComponent.addActionListener(instanceOfMyClass);
   ```

3. Code that implements the methods in the listener interface. For example:
   
   ```java
   public void actionPerformed(ActionEvent e) {
       //code that reacts to the action...
   }
   ```
Let’s look at a typical event-handling scenario of buttons (JButton) handle mouse clicks. To detect when the user clicks an on-screen button (or does the keyboard equivalent), a program must have an object that implements the ActionListener interface. The program must register this object as an action listener on the button (the event source), using the addActionListener method. When the user clicks the on-screen button, the button fires an action event. This results in the invocation of the action listener’s actionPerformed method (the only method in the ActionListener interface). The single argument to this method is an ActionEvent object that gives information about the event and its source.

Event handlers can be instances of any class. Often, an event handler that has only a few lines of code is implemented using an anonymous inner class -- an unnamed class defined inside of another class. Anonymous inner classes can be somewhat confusing at first, but once you’re used to them they make code clearer by keeping the implementation of an event handler close to where the event handler is registered.
17.3. Threads and Event Handling

Event-handling code executes in a single thread, the event-dispatching thread. This ensures that each event handler will finish executing before the next one executes. For instance, the action-Performed method in the preceding example executes in the event-dispatching thread. Painting code also executes in the event-dispatching thread. This means that while the actionPerformed method is executing, the program’s GUI is frozen — it won’t repaint or respond to mouse clicks, for example.

Important: The code in event handlers should execute very quickly! Otherwise, your program’s perceived performance will be poor. If you need to perform some lengthy operation as the result of an event, do it by starting up another thread (or somehow sending a request to another thread) to perform the operation.
17.4. Interface MouseListener

mouseClicked
   public void mouseClicked(MouseEvent e)
   Invoked when the mouse has been clicked on a component.

mousePressed
   public void mousePressed(MouseEvent e)
   Invoked when a mouse button has been pressed on a component.

mouseReleased
   public void mouseReleased(MouseEvent e)
   Invoked when a mouse button has been released on a component.

mouseEntered
   public void mouseEntered(MouseEvent e)
   Invoked when the mouse enters a component.

mouseExited
   public void mouseExited(MouseEvent e)
   Invoked when the mouse exits a component.
Implementation of the interface:

```java
import javax.swing.*; //This is the final package name.
import java.awt.*;
import java.awt.event.*;

public class MyMouseListener implements MouseListener {

    public void mousePressed(MouseEvent e) {
        System.out.println("mousePressed:");
        System.out.println("\te.getPoint(): " + e.getPoint());
        System.out.println("\te.getClickCount(): "
            + e.getClickCount());
    }

    public void mouseReleased(MouseEvent e) {
        // System.out.println("\t" + e);
    }

    public void mouseEntered(MouseEvent e) {
        System.out.println("mouseEntered:");
        System.out.println("\t" + e);
    }

    public void mouseExited(MouseEvent e) {
        System.out.println("mouseExited:");
        System.out.println("\t" + e);
    }

    public void mouseClicked(MouseEvent e) {
        // System.out.println("\t" + e);
    }
}

Source Code: Src/13/MyMouseListener.java
```
import javax.swing.*; // This is the final package name.
import java.awt.*;
import java.awt.event.*;

public class Events {

    public Component createComponents() {
        JTextField text;
        JPanel pane = new JPanel();

        pane.setBorder(BorderFactory.createLoweredBevelBorder());
        pane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
        text = new JTextField("I’m here");
        text.addMouseListener( new MyMouseListener());

        pane.add(text);
        return pane;
    }

    public static void main(String[] args) {

        String lookAndFeel = UIManager.getCrossPlatformLookAndFeelClassName();

        try {
            UIManager.setLookAndFeel( lookAndFeel);
        } catch (Exception e) { }

        JFrame frame = new JFrame("Events");
        Events app = new Events();
        Component contents = app.createComponent();

        frame.getContentPane().add(contents);
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
        frame.pack();
        frame.setVisible(true);
    }
}

Source Code: Src/13/Events.java
Result:

java Events

mouseEntered:
java/gawt.event.MouseEvent[MOUSE_ENTERED,(23,14),mods=0,
clickCount=0] on javax.swing.JTextField[,31,7,50x21,
alignmentX=null,alignmentY=null,
border=javax.swing.plaf.BorderUIResource$CompoundBorderUIResource@fd70c5,flags=0,maximumSize=,minimumSize=,preferredSize=,
caretColor=javax.swing.plaf.ColorUIResource[r=0,g=0,b=0],
disabledTextColor=javax.swing.plaf.ColorUIResource[r=153,g=153,b=153],
editable=true,margin=javax.swing.plaf.InsetsUIResource
[top=0,left=0,bottom=0,right=0],opaque=true,
selectedTextColor=javax.swing.plaf.ColorUIResource[r=0,g=0,b=0],
selectionColor=javax.swing.plaf.ColorUIResource
[r=204,g=204,b=255],columns=0,columnWidth=0,command=,
horizontalAlignment=LEFT]

mousePressed:
  e.getPoint(): java/gawt.Point[x=19,y=12]
  e.getClickCount(): 1

mouseReleases:
mouseClicked:
mousePressed:
  e.getPoint(): java/gawt.Point[x=20,y=11]
  e.getClickCount(): 1

mouseReleases:
mouseClicked:
17.5. Availability

Thus, every component supports component, focus, key, mouse, and mouse-motion listeners. However, a component fires only those events for which listeners have registered on it. For example, if a mouse listener is registered on a particular component, but the component has no other listeners, then the component will fire only mouse events — no component, focus, key, or mouse-motion events.

Listeners supported by Swing components fall into two categories:

- Listeners that All Swing Components Support
- Other Listeners that Swing Components Support

Listeners that All Swing Components Support

Because all Swing components descend from the AWT Component class, you can register the following listeners on any Swing component:

- **component listener**
  
  Listens for changes in the component’s size, position, or visibility.

- **focus listener**
  
  Listens for whether the component gained or lost the ability to receive keyboard input.

- **key listener**
  
  Listens for key presses; key events are fired only by the component that has the current keyboard focus.

- **mouse events**
  
  Listens for mouse clicks and mouse movement into or out of the component’s drawing area.

- **mouse-motion events**
  
  Listens for changes in the cursor’s position over the component.

Other Listeners that Swing Components Support

The following table lists Swing components and the listeners that they support. In many cases, the events are fired directly from the component. In other cases, the events are fired from the component’s data or selection model. To find out the details for the particular component and listener you’re interested in, go first to the component how-to section, and then if necessary to the listener how-to section.
17.6. Checkbox

```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class CheckBoxDemo extends JPanel {
  JCheckBox chinButton;
  JCheckBox glassesButton;
  JCheckBox hairButton;
  JCheckBox teethButton;

  /*
   * Four accessory choices provide for 16 different
   * combinations. The image for each combination is
   * contained in a separate image file whose name indicates
   * the accessories. The filenames are "geek-XXXX.gif"
   * where XXXX can be one of the following 16 choices.
   * The "choices" StringBuffer contains the string that
   * indicates the current selection and is used to generate
   * the file name of the image to display.
   *
   ---- // zero accessories
   c--- // one accessory
   -g--
   --h-
   ---t
   cg-- // two accessories
   c-h-
   c--t
   -gh-
   -g-t
   --ht
   -ght // three accessories
   c-ht
   cg-t
   cgh-
   cght // all accessories
   */

  StringBuffer choices;
  JLabel pictureLabel;

  public CheckBoxDemo() {
    // Create the check boxes
    chinButton = new JCheckBox("Chin");
    chinButton.setMnemonic(KeyEvent.VK_C);
    chinButton.setSelected(true);
  }
```
glassesButton = new JCheckBox("Glasses");
glassesButton.setMnemonic(KeyEvent.VK_G);
glassesButton.setSelected(true);

hairButton = new JCheckBox("Hair");
hairButton.setMnemonic(KeyEvent.VK_H);
hairButton.setSelected(true);

teethButton = new JCheckBox("Teeth");
teethButton.setMnemonic(KeyEvent.VK_T);
teethButton.setSelected(true);

// Register a listener for the check boxes.
CheckBoxListener myListener = new CheckBoxListener();
chinButton.addItemListener(myListener);
glassesButton.addItemListener(myListener);
hairButton.addItemListener(myListener);
teethButton.addItemListener(myListener);

// Indicates what’s on the geek.
choices = new StringBuffer("cght");

// Set up the picture label
pictureLabel = new JLabel(new ImageIcon(
    "images/geek-" + choices.toString() + ".gif"));
pictureLabel.setToolTipText(choices.toString());

// Put the check boxes in a column in a panel
JPanel checkPanel = new JPanel();
checkPanel.setLayout(new GridLayout(0, 1));
checkPanel.add(chinButton);
checkPanel.add(glassesButton);
checkPanel.add(hairButton);
checkPanel.add(teethButton);

setLayout(new BorderLayout());
add(checkPanel, BorderLayout.WEST);
add(pictureLabel, BorderLayout.CENTER);
setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

/** Listens to the check boxes. */
class CheckBoxListener implements ItemListener {
    public void itemStateChanged(ItemEvent e) {
        int index = 0;
        char c = ‘-’;
        Object source = e.getItemSelectable();

        if (source == chinButton) {
            index = 0;
        }
}
c = 'c';
} else if (source == glassesButton) {
    index = 1;
    c = 'g';
} else if (source == hairButton) {
    index = 2;
    c = 'h';
} else if (source == teethButton) {
    index = 3;
    c = 't';
}

if (e.getStateChange() == ItemEvent.DESELECTED)
    c = '-';

choices.setCharAt(index, c);
pictureLabel.setIcon(new ImageIcon{
    "images/geek-
    + choices.toString()
    + ".gif"));
pictureLabel.setToolTipText(choices.toString());
}

public static void main(String s[]) {
    JFrame frame = new JFrame("CheckBoxDemo");
    frame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
    frame.setContentPane(new CheckBoxDemo());
    frame.pack();
    frame.setVisible(true);
}

Source Code: Src/13/CheckBoxDemo.java
17.7. Which One?

Result:

mouseEntered:
  e.getSource(): bbbbbbbbbbbbbbbbbbbbbbbbbbbbb
mouseExited:
  e.getSource(): bbbbbbbbbbbbbbbbbbbbbbbbbbbbb
mouseEntered:
  e.getSource(): bbbbbbbbbbbbbbbbbbbbbbbbbbbbb
mousePressed:
  e.getSource(): bbbbbbbbbbbbbbbbbbbbbbbbbbbbb
mouseReleases:
  e.getSource(): bbbbbbbbbbbbbbbbbbbbbbbbbbbbb
mouseClicked:
  e.getSource(): bbbbbbbbbbbbbbbbbbbbbbbbbbbbb
mouseExited:
  e.getSource(): bbbbbbbbbbbbbbbbbbbbbbbbbbbbb
mouseEntered:
  e.getSource(): aaaaaaaaaa
mousePressed:
  e.getSource(): aaaaaaaaa
import javax.swing.*; //This is the final package name.
import java.awt.*;
import java.awt.event.*;

public class WhichOne {

    public Component createComponents() {
        JButton b1;
        JButton b2;
        GridBagLayout gridbag = new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();
        JPanel contentPane = new JPanel();

        c.gridwidth = 2;
        c.gridy = 0;
        c.gridx = 0;
        contentPane.setLayout(gridbag);
        b1 = new JButton("aaaaaaaaa");
        b1.addMouseListener(new MyMouseListener2());
        gridbag.setConstraints(b1, c);
        contentPane.add(b1);

        c.gridwidth = 2;
        c.gridy = 0;
        c.gridx = 1;
        contentPane.setLayout(gridbag);
        b2 = new JButton("bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb");
        b2.addMouseListener(new MyMouseListener2());
        gridbag.setConstraints(b2, c);
        contentPane.add(b2);

        return contentPane;
    }

    public static void main(String[] args) {
        String lookAndFeel = UIManager.getCrossPlatformLookAndFeelClassName();

        try {
            UIManager.setLookAndFeel(lookAndFeel);
        } catch (Exception e) {
        }

        JFrame frame = new JFrame("WhichOne");
        WhichOne app = new WhichOne();
        Component contents = app.createComponent();

        frame.getContentPane().add(contents);
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }
}
54         }
55     });
56     frame.pack();
57     frame.setVisible(true);
58 }
59 }

Source Code: Src/13/WhichOne.java
17.8. Improved Lotto Handler ...

```java
import javax.swing.*;  //This is the final package name.
import java.awt.*;
import java.awt.event.*;

public class LottoE {

    public void createButtons( GridBagLayout gridbag,
                JPanel contentPane,
                int gridy) {
        int number = 1;

        GridBagConstraints c = new GridBagConstraints();
        c.gridwidth = 1;
        c.weightx = 1.0;
        c.insets = new Insets(5,3,3,5);
        c.fill = GridBagConstraints.BOTH;

        ActionListener aActionListener =
                new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JButton o = (JButton)e.getSource();
                System.out.println("getText = " +
                        o.getText());
                if (o.getBackground() == Color.red)
                        o.setBackground(Color.yellow);
                else
                        o.setBackground(Color.red);
            }
        };

        for (int col = 1; col <= 7; col++) {
            for (int row = 1; row <= 7; row++) {
                JButton button = new JButton( new Integer(number++).toString());
                button.setBackground(Color.yellow);
                button.setMinimumSize( button.getSize() );
                button.addActionListener( aActionListener);
                c.gridx = row ;
                c.gridy = col + gridy;
                gridbag.setConstraints(button, c);
                contentPane.add(button);
            }
        }
    }

    public void createSeparator( GridBagLayout gridbag,
                JPanel contentPane,
                int gridy) {
        GridBagConstraints c = new GridBagConstraints();
```
JSeparator sep = new JSeparator(SwingConstants.HORIZONTAL);

c.fill = GridBagConstraints.BOTH;
c.gridx = 0;
c.gridy = gridy;
c.gridwidth = 8;
c.insets = new Insets(5,3,3,5);
gridbag.setConstraints(sep, c);
contentPane.add(sep);
}

public Component createComponents() {
    JSeparator sep;
    JButton button;
    JLabel label;
    JPanel contentPane = new JPanel();
    GridBagLayout gridbag = new GridBagLayout();
    GridBagConstraints c = new GridBagConstraints();
    contentPane.setLayout(gridbag);

    label = new JLabel("0");
c.weightx = 0.5;
c.gridx = 0;
c.gridy = 0;
c.gridwidth = 8;
c.ipady = 10;
gridbag.setConstraints(label, c);
contentPane.add(label);

    createSeparator(gridbag, contentPane, 1);
    createButtons(gridbag, contentPane, 3);
    createSeparator(gridbag, contentPane, 11);

    JLabel label_1 = new JLabel("0 correct");
c.weightx = 0.5;
c.gridx = 0;
c.gridy = 12;
c.gridwidth = 4;
c.ipady = 10;
gridbag.setConstraints(label_1, c);
contentPane.add(label_1);

    JTextField tField = new JTextField("0", 5);
tField.setHorizontalAlignment(JTextField.RIGHT);
tField.setEditable(false);
c.weightx = 0.5;
c.gridx = 4;
c.gridy = 12;
c.gridwidth = 4;
c.ipady = 10;
gridbag.setConstraints(tField, c);
contentPane.add(tField);

    createSeparator(gridbag, contentPane, 13);
}
return contentPane;

public static void main(String[] args) {
    String lookAndFeel = UIManager.getCrossPlatformLookAndFeelClassName();
    try {
        UIManager.setLookAndFeel(lookAndFeel);
    } catch (Exception e) { }
    JFrame frame = new JFrame("LottoE");
    LottoE app = new LottoE();
    Component contents = app.createComponent();
    frame.getContentPane().add(contents);

    //Finish setting up the frame, and show it.
    frame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
    frame.pack();
    frame.setVisible(true);
}

Source Code: Src/13/LottoE.java
17.9. An Interesting Example: The App

17.10. Using DefaultStyledDocument

Stolen and modified from:

- Like other Swing components, a text component separates its data (known as the model) from its view of the data.
- A text component’s model is known as a document and is an instance of a class that implements the Document interface. A document provides these services for a text component:
  - Contains the text. A document stores the textual content in Element objects, which can represent any logical text structure, such as paragraphs, text runs that share styles, and so on. We do not cover Elements. However, The Swing Connection has at least one article on the subject.
  - Provides support for editing the text through the remove and insertString methods.
  - Notifies document listeners and undoable edit listeners of changes to the text.
  - Manages Position objects, which track a particular location within the text even as the text is modified.
  - Allows you to get information about the text, such as its length, and segments of the text as a string.

See also:

17.11. Extreme Editor: Interface

```java
//****************************************************************************
// File: HpExtremeInterface.java
//****************************************************************************
import javax.swing.text.AttributeSet;
import java.awt.Color;

/**
 * Class HpExtremeInterface is the interface for the extreme editor pane.
 * It allows to insert a string at a given location,
 * it allows to remove a string from a given location, and
 * removes the complete content.
 *
 * <P>
 * @author Hans-Peter Bischof
 * @version October/20/2003
 */

public interface HpExtremeInterface {

/**
 * Inserts text in the editor window
 */
```
/**
 * @param sender originated the change
 * @param offs offset
 * @param aColor used color
 */
public void insertString(HpExtremeInterface sender,
int offs,
String str,
Color aColor); // should be AttributeSet

/**
 * removes text from the editor window
 * @param sender originated the change
 * @param offs offset
 * @param changeLength so many chars
 */
public void remove(HpExtremeInterface sender,
int offSet,
int changeLength);

/**
 * removes all text from the editor window
 * @param sender originated the change
 */
public void removeContent(HpExtremeInterface sender);

Src/13e/HpExtremeInterface.java

17.12. Extreme Editor: StyledDocument

//****************************************************************************
// File: HpStyledDocument.java
//*****************************************************************************
import javax.swing.text.DefaultStyledDocument;
import javax.swing.text.AttributeSet;
import javax.swing.text.BadLocationException;
import javax.swing.text.MutableAttributeSet;
import javax.swing.text.StyleConstants;
import javax.swing.text.SimpleAttributeSet;
import java.awt.Color;
import edu.rit.m2mi.M2MI;
import edu.rit.m2mi.SynthesisException;
import java.util.Enumeration;
import java.io.ByteArrayOutputStream;
import java.io.ObjectOutputStream;

/**
 * Class HpStyledDocument implements HpExtremeInterface and
 * is the functional part for the extreme editor.
 * Synchronization is not implemented.
 * This should be done.
 * <P>
 * <B>M2MI Library Version October/20/2003</B>
 * <P>
 * @author Hans-Peter Bischof
 * @version October/20/2003
 */

public class HpStyledDocument extends DefaultStyledDocument
    implements HpExtremeInterface {

    protected HpExtremeInterface allEditors = null;    // Omnihandle
    protected HpExtremeInterface localEditor = null;    // uniHandle
    protected ExtremeEditor extremEditor = null;        // the editor
    protected static AttributeSet sA;                   // attribute Set

    /**
     * Creates a new Document.
     * @param extremEditor the editor in the window
     */
    public HpStyledDocument(ExtremeEditor extremEditor) {
        this.extremEditor = extremEditor;
        try {
            allEditors = (HpExtremeInterface)
                M2MI.getOmnihandle (HpExtremeInterface.class);
        } catch (Throwable exc) {
            exc.printStackTrace (System.err);
        }
    }

    /**
     * Exports the editor into the m2mi layer
     * @return the exported editor
     */
    public HpStyledDocument export() {
        try {
            M2MI.export (this, HpExtremeInterface.class);
            localEditor = ((HpExtremeInterface)
                M2MI.getUnihandle(this, HpExtremeInterface.class));
            return this;
        } catch (Throwable exc) {
            System.err.println ("HpStyledDocument: Uncaught exception";
            exc.printStackTrace (System.err);
        }
    }
}
private void setTitle()
{
    return this;
}

/**
 * Inserts text in the editor window
 * @param sender originated the change
 * @param offs offset
 * @param a used attribute set
 */
public void insertString(int offs, String str, AttributeSet a) // overwrites ....
throws BadLocationException {
try {
    sA = a;
    Color aColor = extremEditor.getColor();
    MutableAttributeSet attr = null;
    if (aColor != null) {
        attr = new SimpleAttributeSet();
        StyleConstants.setForeground(attr, aColor);
    }
    super.insertString(offs, str, attr); /// why super?
    allEditors.insertString(localEditor, offs, str, extremEditor.getColor());
} catch (Throwable exc) {
    exc.printStackTrace (System.err);
}
}

/**
 * Inserts text in the editor window
 * @param sender originated the change
 * @param offs offset
 * @param aColor used Color should be attribute set
 */
public void insertString(HpExtremeInterface sender, int offs, String str, Color aColor) {
    MutableAttributeSet attr = null;
    if (aColor != null) {
        attr = new SimpleAttributeSet();
        StyleConstants.setForeground(attr, aColor);
    }
    try {
        if (!this.localEditor.equals(sender)) {
            super.insertString(offs, str, attr);
        }
    } catch (Throwable exc) {
        exc.printStackTrace (System.err);
    }
}

/**
public void remove(int offset, int changeLength) {
    try {
        super.remove(offset, changeLength);
        allEditors.remove(localEditor, offset, changeLength);
    } catch (Throwable exc) {
        exc.printStackTrace(System.err);
    }
}

/**
 * Removes all content
 * Removes the local content first, then sends out
 * a remove content method to all extreme editor objects.
 */
public void removeContent() {
    try {
        int length = getLength();
        super.remove(0, length);
        allEditors.removeContent(localEditor);
    } catch (Throwable exc) {
        exc.printStackTrace(System.err);
    }
}

/*
 * Remove all content if the sender is not the local editor
 */
public void removeContent(HpExtremeInterface sender) {
    try {
        if (!this.localEditor.equals(sender)) {
            super.remove(0, length);
            allEditors.removeContent(localEditor);
        }
    } catch (Throwable exc) {
        exc.printStackTrace(System.err);
    }
}
int length = getLength();
    super.remove(0, length);
}
} catch (Throwable exc) {
    exc.printStackTrace (System.err);
}
}

Source Code: Src/13e/HpStyledDocument.java

17.13. Extreme Editor: ExtremeEditor

import javax.swing.JFrame;
import java.awt.Component;
import java.awt.Container;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.InputMethodListener;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Color;
import javax.swing.JScrollPane;
import javax.swing.text.DefaultStyledDocument;
import javax.swing.JTextPane;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JMenu;
import javax.swing.JColorChooser;
import javax.swing.text.MutableAttributeSet;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.StyleConstants;
import javax.swing.text.StyledEditorKit;
import javax.swing.text.AttributeSet;
import javax.swing.text.StyledDocument;
import javax.swing.JFileChooser;
import javax.swing.ImageIcon;
import javax.swing.Icon;
import javax.swing.JOptionPane;
import java.util.Hashtable;
import java.io.File;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.FileWriter;
import java.io.BufferedWriter;

import edu.rit.m2mi.M2MI;
import edu.rit.m2mi.SynthesisException;

/**
 * Class ExtremeEditor is the driver of the extreme editor.
 * Order of execution:
 * <UL>
 * <UL>
 * Initialize M2MI
 * <UL>
 * paring the arguments --> split the frame in two panes or not
 * <UL>
 * adding a HpStyledDocument to the pane
 * <UL>
 * </UL>
 * <P>
 * <B>M2MI Library Version October/20/2003</B>
 * @author Hans-Peter Bischof
 * @version October/20/2003
 */

public class ExtremeEditor extends JFrame {

    /**
     * Initialize the M2MI layer.
     */
    static {
        try {
            M2MI.initialize(54321L);
        } catch (Throwable exc) {
            System.err.println("ExtremeEditor: Uncaught exception");
            exc.printStackTrace(System.err);
        }
    }

    final protected static int WIDTH = 500; // with of the frame
    final protected static int HEIGHT = 500; // height of the frame
    static protected int HEIGHT_F = HEIGHT; // height if split
    final protected static int MAX_READ = 512; // # chars read at once
    protected JTextPane pane = null; // editor
    protected JTextPane ccPane = null; // carbon copy editor
    protected Hashtable aHashTable = new Hashtable(); // for colors
    protected JMenuBar menueBar = new JMenuBar(); // menu bar
    protected Color aColor = null; // actual color
    protected boolean split = false; // see parseArgs

    /**
* Creates a new ExtremeEditor.
* @param text Will be used to set the titlebar.
* */
public ExtremeEditor(String text) {
    super(text);
}

/**
* Returns the current used foreground color
* @return the current used foreground color
*/
public Color getColor() {
    return aColor;
}

/**
* Allows to choose a new foreground color.
*/
public void doColorCommand() {
    aColor = JColorChooser.showDialog(
        this, "Color Chooser", Color.cyan);
    if (aColor != null) {
        MutableAttributeSet attr = new SimpleAttributeSet();
        StyleConstants.setForeground(attr, aColor);
        pane.setCharacterAttributes(attr, false);
    }
}

 /**
* Loads a new file into the current buffer.
* A modal dialog window will open if the file does not exist.
*/
public void doLoadCommand() {
    JFileChooser chooser = new JFileChooser();
    int status = chooser.showOpenDialog(this);
    if (status == JFileChooser.APPROVE_OPTION) {
        File file = chooser.getSelectedFile();
        try {
            HpStyledDocument doc = (HpStyledDocument)pane.getStyledDocument();
            doc.removeContent();
            FileReader fileInputReader = new FileReader(file);
            BufferedReader bufferedReader = new BufferedReader(fileInputReader);
            char buffer[] = new char[MAX_READ];
            int len;
            while ((len = bufferedReader.read(buffer, 0, buffer.length)) != -1) {
                doc.insertString(doc.getLength(),
                    new String(buffer, 0, len), null);
            }
        }
    }
}
### Public Methods

#### `doSaveCommand()`

```java
public void doSaveCommand() {

    JFileChooser chooser = new JFileChooser();
    int status = chooser.showSaveDialog(this);
    if (status == JFileChooser.APPROVE_OPTION) {
        File file = chooser.getSelectedFile();
        try {
            HpStyledDocument doc = (HpStyledDocument)pane.getStyledDocument();
            FileWriter fileOutputWriter = new FileWriter (file);
            BufferedWriter bufferedWriter =
                new BufferedWriter(fileOutputWriter);
            int length = doc.getLength();
            bufferedWriter.write( doc.getText(0, length) );
            bufferedWriter.flush();
            bufferedWriter.close();
        } catch (Exception exc) {
            JOptionPane.showMessageDialog(null,
                "Problems with Parsing the file", "alert",
                JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

#### `createFileMenue(JMenuBar menueBar)`

```java
private void createFileMenue(JMenuBar menueBar) {
    JMenu file = new JMenu ("File");
    JMenuItem item;
    file.add (item = new JMenuItem ("Load"));
    item.addActionListener (new ActionListener() {
        public void actionPerformed (ActionEvent e) {
            doLoadCommand();
        }
    });
    file.add (item = new JMenuItem ("Save"));
    item.addActionListener (new ActionListener() {
        public void actionPerformed (ActionEvent e) {
```
doSaveCommand();

file.addSeparator();
file.add (item = new JMenuItem("Close"));
menueBar.add (file);
item.addActionListener (new ActionListener () {
    public void actionPerformed (ActionEvent e) {
        System.exit(0);
    }
});

ActionListener aActionListener =
    new ActionListener () {
        public void actionPerformed (ActionEvent e) {
            JMenuItem o = (JMenuItem)e.getSource();
            MutableAttributeSet attr = new SimpleAttributeSet();
            aColor = (Color)aHashTable.get(o.getText());
            if (aColor != null) { // can not happen!
                StyleConstants.setForeground(attr, aColor);
                pane.setCharacterAttributes(attr, false);
            }
        }
    };

/**
 * Allows to pick a new foreground color.
 * This color will be used for all write activities.
 *
 * @param label the label
 * @param aColor printed in this forground color
 * @return the current used foreground color
 */
private JMenuItem addColorItem(String label, Color aColor) {
    JMenuItem item = new JMenuItem(label);
    aHashTable.put(label, aColor);
    item.setIcon(new ColorSquare(aColor));
    item.addActionListener(aActionListener);
    return item;
}

/**
 * Creates a new Color Menue
 *
 * @param menueBar under this menueBar
 */
private void createColorMenue(JMenuBar menueBar) {
    JMenuItem item;
    JMenu color = new JMenu("Color");
    color.add (addColorItem("Red", Color.red));
    color.add (addColorItem("Blue", Color.blue));
color.add (addColorItem("Green", Color.green));

color.add (item = new JMenuItem ("Custom Color");
item.addActionListener (new ActionListener() {
    public void actionPerformed (ActionEvent e) {
        doColorCommand();
    }
});
menueBar.add (color);

/**
 * Creates a the the menueBar and the menues
 *
 */
private void createMenues() {
    JMenuBar menueBar = new JMenuBar();
    setJMenuBar (menueBar);

    createFileMenue(menueBar);
    createColorMenue(menueBar);
}

/**
 * Creates all the visible elements
 * Two HpStyledDocument will appear in the frame if -split
 * is an argument. This is more for testing purpose.
 */
private Component createFrameComponents() {
    Container aContainerPane = getContentPane();

    pane = new JTextPane( ( new HpStyledDocument(this)).export());
    JScrollPane sp = new JScrollPane (pane);
    if (split) {
        ccPane = new JTextPane( ( new HpStyledDocument(this)).export());
        HEIGHT_F = (int)( HEIGHT * 1.2);
        sp.setPreferredSize(new Dimension(WIDTH, HEIGHT / 2 ));
        JScrollPane ccSp = new JScrollPane (ccPane);
        ccSp.setPreferredSize(new Dimension(WIDTH, HEIGHT / 2 ));
        aContainerPane.add(ccSp, BorderLayout.SOUTH);
    } else
        sp.setPreferredSize(new Dimension(WIDTH, HEIGHT ));
    aContainerPane.add(sp, BorderLayout.NORTH);
}

return aContainerPane;

/**
 * Print the usage message and terminates.
 */
11 private void printMessage() {
12     System.out.println("-h ----> help");
13     System.out.println("[-split] ----> split the screen");
14     System.exit(0);
15 }
16
17 /**
18 * Parse the command line arguments and sets variables.
19 * The program terminates, if the arguments include -h or
20 * if one argument can not be recognized.
21 */
22 public void parseArgs(String args[]) {
23     for (int i = 0; i < args.length; i++) {
24         if (args[i].equals("-h"))
25             printMessage();
26         else if (args[i].equals("-split")) {
27             split = true;
28         } else
29             printMessage();
30     }
31 }
32
33 /**
34 * Main program
35 */
36 public static void main(String[] args) {
37     ExtremeEditor app = new ExtremeEditor("ExtremeEditor");
38     app.parseArgs(args);
39     Component contents = app.createFrameComponents();
40     app.createMenues();
41     app.addWindowListener(new WindowAdapter() {
42         public void windowClosing(WindowEvent e) {
43             System.exit(0);
44         }
45     });
46     app.setSize(WIDTH, HEIGHT_F);
47     app.setVisible(true);
48 }
49
50 Source Code: Src/13e/ExtremeEditor.java

17.14. Extreme Editor: Colors
import java.awt.Color;
import java.awt.Component;
import java.awt.Graphics;
import javax.swing.Icon;

class ColorSquare implements Icon {
    private Color color; // this color will be used
    final static int WIDTH = 7; // size of the square

    public ColorSquare(Color color) {
        this.color = color;
    }

    public void paintIcon (Component c, Graphics g, int x, int y) {
        g.setColor(color);
        g.fillRect (1, 1, WIDTH-1, WIDTH-1);
    }

    public int getIconWidth() {
        return WIDTH;
    }
}

/**
 * Class ColorSquare allows to paint colored icons in a menu.
 *<P>
 *<B>M2MI Library Version October/20/2003</B>
 *<P>
 *@author Hans-Peter Bischof
 *<P>
 *@version October/20/2003
 */
55     public int getIconWidth() {
56         return WIDTH;
57     }
58
59     /**
60      * returns the height of the icon
61      *
62      * @return the height
63      */
64     public int getIconHeight() {
65         return WIDTH;
66     }
67 }

Source Code: Src/13e/ColorSquare.java

17.15. Extreme Editor: Makefile

1    CLASSPATH=$(HOME)/Anhinga/lib/m2mi/lib:.
2    JAVA=java -classpath $(CLASSPATH)
3
4    go: ColorSquare.class ExtremeEditor.class HpExtremeInterface.class \
5        HpStyledDocument.class
6        $(JAVA) ExtremeEditor -split
7
8    doc:
9        javadoc -overview overview.html \   
10       -d javadoc \   
11       -use \   
12       -splitIndex \   
13       -windowtitle 'Extreme Editor' \   
14       -doctitle 'Anhinga - <sup><font size="-2">TM</font></sup><font size="-1">v1.0</font>' \   
15       -header '<b>Anhinga</b> <br><font size="-1">v1.0</font>' \   
16       -bottom 'Copyright 2003 hpb' \   
17       -version \   
18       -author *.java
19
20    HpDocumentListener.class: HpDocumentListener.java
21        $(JAVA) HpDocumentListener.java
22
23    rm:
24    rm -f *.class

Source Code: Src/13e/makefile

18. Painting

When a Swing GUI needs to paint itself -- whether for the first time, in response to becoming
unhidden, or because it needs to reflect a change in the program’s state -- it starts with the high-
est component that needs to be repainted and works its way down the containment hierarchy.
This process is orchestrated by the AWT painting system, and made more efficient and smooth
by the Swing repaint manager and double-buffering code.
Swing components generally repaint themselves whenever necessary. When you invoke the `setText` method on a component, for example, the component should automatically repaint itself and, if appropriate, resize itself. If it doesn’t, it’s a bug. The workaround is to invoke the repaint method on the component to request that the component be scheduled for painting. If the component’s size or position needs to change but doesn’t do so automatically, you should invoke `revalidate` upon the component before invoking `repaint`.

Like event-handling code, painting code executes on the event-dispatching thread. While an event is being handled, no painting will occur. Similarly, if a painting operation takes a long time, no events will be handled during that time.

Programs should paint only when the painting system tells them to because each occurrence of a component painting itself must execute without interruption. Otherwise, unpredictable results could occur, such as a button being painted as half pressed and half unpressed.

For smoothness, Swing painting is double-buffered by default -- performed to an offscreen buffer and then flushed to the screen once finished. It might slightly help performance if you make a Swing component opaque, so that the Swing painting system can know not to paint anything behind the component. To make a Swing component opaque, invoke `setOpaque(true)` on the component.

Although their available painting area is always rectangular, non-opaque Swing components can appear to be any shape. A button, for instance, might display itself by painting a filled octagon. The component behind the button (its container, most likely) would then be visible, showing through at the corners of the button’s bounds. The button would have to include special hit detection code to avoid acting pressed if the user happens to click on its corners.
18.1. An Example of Painting

To illustrate painting, we'll use the following hierarchy:

line from 1.250,5.500 to 1.250,6.125 to 3.750,6.125 to 3.750,5.500 line from 2.500,6.125 to 2.500,6.750 line from 2.500,7.125 to 2.500,7.750 line from 2.500,8.250 to 2.500,8.875 "JButton" at 1.000,5.289 ljust "JLabel" at 3.438,5.289 ljust "JPanel" at 2.125,6.914 ljust "JFrame" at 2.250,9.039 ljust "content pane" at 2.125,7.914 ljust
1. The top-level container, JFrame, paints itself.

2. The content pane first paints its background, which is a solid gray rectangle. It then tells the JPanel to paint itself. The content pane’s background rectangle doesn’t actually appear in the finished GUI because the content pane is completely obscured by the JPanel.

   Note: It’s important that the content pane be opaque. Otherwise, messy repaints will result. Because the JPanel is opaque, we could make it the content pane (by substituting `setContentPane` for the existing code `getContentPane().add`). This would slightly simplify the containment hierarchy and painting by removing an unnecessary container.

3. The JPanel first paints its background, a solid gray rectangle. Next, it paints its border. The border is an EmptyBorder, which has no effect except for increasing the JPanel’s size by reserving some space at the edge of the panel. Finally, the panel asks its children to paint themselves.

4. To paint itself, the JButton paints its background rectangle, if necessary, and then paints the text that it contains. If the button has the keyboard focus, meaning that any typing goes directly to the button for processing, then the button does some look-and-feel-specific painting to make clear that it has the focus.

5. To paint itself, the JLabel paints its text.

In this way, each component paints itself before any of the components it contains. This ensures that the background of a JPanel, for example, is visible only where it isn’t covered by painting performed by one of the components it contains. The following figure illustrates the order in which each component that inherits from JComponent paints itself.
If you really need to perform custom painting, then you need to decide which superclass to use. I recommend that you extend either JPanel or a more specialized Swing component class. For example, if you’re creating a custom button class, you should probably implement it by extending a button class such as JButton or JToggleButton. That way you’ll inherit the state management provided by those classes. If you’re creating a component that paints on top of an image, you might want to create a JLabel subclass. On the other hand, if you’re implementing a component that generates and displays a graph on top of a blank or transparent background, then you might want to use a JPanel subclass.

When implementing custom painting code, keep two things in mind:

• Your custom painting code belongs to a method named paintComponent.
• You can -- and probably should -- use a border to paint the outside edges of your component.
18.2. The Coordinate System

Each component has its own integer coordinate system, ranging from (0, 0) to (width - 1, height - 1), with each unit representing the size of one pixel. As the following figure shows, the upper left corner of a component’s painting area is (0, 0). The X coordinate increases to the right, and the Y coordinate increases downward.

box with .sw at (1.38,7.75) width 2.31 height 1.25 line -> from 0.750,9.750 to 1.375,9.000 line -> from 1.375,9.000 to 4.562,9.000 line -> from 1.375,9.000 to 1.375,7.125 line -> from 4.125,7.125 to 3.688,7.750 "0/0" at 0.562,9.908 ljust "x" at 4.750,9.096 ljust "y" at 1.062,7.158 ljust "component" at 1.938,8.408 ljust "width-1/height-1" at 4.250,7.158 ljust
18.3. Draw a Yellow Rectangle

```java
import javax.swing.*;
import javax.swing.border.Border;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

public class RectangleDemo {
    JLabel label;

    void buildUI(Container container) {
        container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));

        RectangleArea rectangleArea = new RectangleArea(this);
        container.add(rectangleArea);

        label = new JLabel("Click within the framed area.");
        container.add(label);
    }

    public void updateLabel(Point point) {
        label.setText("Click occurred at coordinate (" + point.x + ", " + point.y + ").");
    }

    //Called only when this is run as an application.
    public static void main(String[] args) {
        JFrame f = new JFrame("RectangleDemo");
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
        RectangleDemo controller = new RectangleDemo();
        controller.buildUI(f.getContentPane());
        f.pack();
        f.setVisible(true);
    }
}
```
class RectangleArea extends JPanel {
    Point point = null;
    RectangleDemo controller;
    Dimension preferredSize = new Dimension(300,100);
    int rectWidth = 50;
    int rectHeight = 50;

    public RectangleArea(RectangleDemo controller) {
        this.controller = controller;

        Border raisedBevel = BorderFactory.createRaisedBevelBorder();
        Border loweredBevel = BorderFactory.createLoweredBevelBorder();
        Border compound = BorderFactory.createCompoundBorder
        (raisedBevel, loweredBevel);
        setBorder(compound);

        addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                point = e.getPoint();
                repaint();
            }
        });

        public Dimension getPreferredSize() {
            return preferredSize;
        }

        public void paintComponent(Graphics g) {
            super.paintComponent(g); //paint background

            if (point != null) {
                g.drawRect(point.x, point.y,
                rectWidth - 1, rectHeight - 1);
                g.setColor(Color.yellow);
                g.fillRect(point.x + 1, point.y + 1,
                rectWidth - 2, rectHeight - 2);

                controller.updateLabel(point);
            }
        }
    }
}

Source Code: Src/14/RectangleDemo.java
18.4. Draw a Red/Green/Blue Rectangle

```java
/*
 * Swing version.
 */

import javax.swing.*;
import javax.swing.border.Border;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

public class RedBlueGreen {
    JLabel label;

    void buildUI(Container container) {
        container.setLayout(new BoxLayout(container,
            BoxLayout.Y_AXIS));

        RectangleAreaRBG rectangleArea = new RectangleAreaRBG(this);
        container.add(rectangleArea);

        label = new JLabel("Click within the framed area.");
        container.add(label);
    }

    public void updateLabel(Point point) {
        label.setText("Click occurred at coordinate (" +
                      point.x + ", " + point.y + ").");
    }

    //Called only when this is run as an application.
    public static void main(String[] args) {
        JFrame f = new JFrame("RedBlueGreen");
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
        RedBlueGreen controller = new RedBlueGreen();
        controller.buildUI(f.getContentPane());
        f.pack();
        f.setVisible(true);
    }
}

class RectangleAreaRBG extends JPanel {
    Point point = null;
    Color aColor = Color.red;
    RedBlueGreen controller;
    Dimension preferredSize = new Dimension(300,100);
    int rectWidth = 50;
    int rectHeight = 50;
```
public RectangleAreaRBG(RedBlueGreen controller) {
    this.controller = controller;

    Border raisedBevel = BorderFactory.createRaisedBevelBorder();
    Border loweredBevel = BorderFactory.createLoweredBevelBorder();
    Border compound = BorderFactory.createCompoundBorder
         (raisedBevel, loweredBevel);
    setBorder(compound);

    addMouseListener(new MouseAdapter() {
        public void mousePressed(MouseEvent e) {
            int modifiers = e.getModifiers();
            point = e.getPoint();

            if ((modifiers & InputEvent.BUTTON1_MASK) == InputEvent.BUTTON1_MASK)
                aColor = Color.red;
            else if ((modifiers & InputEvent.BUTTON2_MASK) == InputEvent.BUTTON2_MASK)
                aColor = Color.green;
            else if ((modifiers & InputEvent.BUTTON3_MASK) == InputEvent.BUTTON3_MASK)
                aColor = Color.blue;
            repaint();
        }
    });

    public Dimension getPreferredSize() {
        return preferredSize;
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g); // paint background

        if (point != null) {
            g.drawRect(point.x, point.y,
                rectWidth - 1, rectHeight - 1);
            g.setColor(aColor);
            g.fillRect(point.x + 1, point.y + 1,
                rectWidth - 2, rectHeight - 2);

            controller.updateLabel(point);
        }
    }
}

Source Code: Src/14/RedBlueGreen.java
18.5. Draw many Red/Green/Blue Rectangles

```java
/*
 * Swing version.
 */

import javax.swing.*;
import javax.swing.border.Border;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

public class Many {
    JLabel label;
    
    void buildUI(Container container) {
        container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
        RectangleAreaMany rectangleArea = new RectangleAreaMany(this);
        container.add(rectangleArea);
        label = new JLabel("Click within the framed area.");
        container.add(label);
    }
    
    public void updateLabel(Point point) {
        label.setText("Click occurred at coordinate (" + point.x + ", " + point.y + ").");
    }
    
    // Called only when this is run as an application.
    public static void main(String[] args) {
        JFrame f = new JFrame("Many");
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
        Many controller = new Many();
        controller.buildUI(f.getContentPane());
        f.pack();
        f.setVisible(true);
    }
}

class RectangleAreaMany extends JPanel {
    Point point = null;
    Color aColor = Color.red;
    Many controller;
    Dimension preferredSize = new Dimension(300,100);
    int rectWidth = 50;
    int rectHeight = 50;
```
public RectangleAreaMany(Many controller) {
    this.controller = controller;
    Border raisedBevel = BorderFactory.createRaisedBevelBorder();
    Border loweredBevel = BorderFactory.createLoweredBevelBorder();
    Border compound = BorderFactory.createCompoundBorder
        (raisedBevel, loweredBevel);
    setBorder(compound);

    addMouseListener(new MouseAdapter() {
        public void mousePressed(MouseEvent e) {
            int modifiers = e.getModifiers();
            point = e.getPoint();

            if ( (modifiers & InputEvent.BUTTON1_MASK )
                == InputEvent.BUTTON1_MASK )
                aColor = Color.red;
            else if ( (modifiers & InputEvent.BUTTON2_MASK )
                == InputEvent.BUTTON2_MASK )
                aColor = Color.green;
            else if ( (modifiers & InputEvent.BUTTON3_MASK )
                == InputEvent.BUTTON3_MASK )
                aColor = Color.blue;
            repaint(e.getX(), e.getY(),
                rectWidth, rectHeight);
        }
    });
}

public Dimension getPreferredSize() {
    return preferredSize;
}

public void paintComponent(Graphics g) {
    super.paintComponent(g); //paint background

    if (point != null) {
        g.drawRect(point.x, point.y,
            rectWidth - 1, rectHeight - 1);
        g.setColor(aColor);
        g.fillRect(point.x + 1, point.y + 1,
            rectWidth - 2, rectHeight - 2);
        controller.updateLabel(point);
    }
}

Source Code: Src/14/Many.java
public class Sketch {
    JLabel label;
    JFrame aJFrame;

    public Sketch(String args[], JFrame aJFrame) {
        this.aJFrame = aJFrame;
    }

    void buildUI(Container container) {
        container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
        RectangleAreaSketch rectangleArea = new RectangleAreaSketch(this);
        container.add(rectangleArea);
        label = new JLabel("Click within the framed area.");
        container.add(label);
    }

    public void updateLabel(Point point) {
        label.setText("Click occurred at coordinate (" + point.x + ", " + point.y + ").");
    }

    // Called only when this is run as an application.
    public static void main(String[] args) {
        JFrame f = new JFrame("Sketch");
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
        Sketch aSketch = new Sketch(args, f);
        aSketch.buildUI(f.getContentPane());
        f.pack();
        f.setVisible(true);
    }
}
class RectangleAreaSketch extends JPanel {
    Point point = null;
    Color aColor = Color.red;
    Sketch controller;
    Dimension preferredSize = new Dimension(450,300);

    public RectangleAreaSketch(Sketch controller) {
        this.controller = controller;
        Border raisedBevel = BorderFactory.createRaisedBevelBorder();
        Border loweredBevel = BorderFactory.createLoweredBevelBorder();
        Border compound = BorderFactory.createCompoundBorder
            (raisedBevel, loweredBevel);
        setBorder(compound);
        controller.aJFrame.setTitle("hpb's Sketch Program ... Good Luck");
        addMouseMotionListener(new MouseMotionListener() {
            public void mouseMoved(MouseEvent e) {}
            public void mouseDragged(MouseEvent e) {
                int modifiers = e.getModifiers();
                point = e.getPoint();
                if ((modifiers & InputEvent.BUTTON1_MASK) == InputEvent.BUTTON1_MASK) {
                    aColor = Color.red;
                } else if ((modifiers & InputEvent.BUTTON2_MASK) == InputEvent.BUTTON2_MASK) {
                    aColor = Color.green;
                } else if ((modifiers & InputEvent.BUTTON3_MASK) == InputEvent.BUTTON3_MASK) {
                    aColor = Color.blue;
                }
                repaint(e.getX(), e.getY(), 4, 4);
            }
        });
    }
    public Dimension getPreferredSize() {
        return preferredSize;
    }
    public void paintComponent(Graphics g) {
        String theData = null;
        int diameter = 6;
        int diameterM1 = diameter - 1;
        //super.paintComponent(g); //paint background
        if (point != null) {
            g.drawArc(point.x, point.y, diameter, diameter, 0, 360);
            g.setColor(aColor);
        }
    }
}
18.7. A Ticker Application

- Goal:
- The Ticker class:

```java
public class Ticker extends JPanel implements Runnable {
    int index = 0;
    private long memory = 0;
    private int xPos, yPos;
    private int height, width;
    private int defaultHeight = 40;
    private int defaultWidth = 600;
    private Thread runner = null;
    private String text = null;
    private String runningText = null;
    private FontMetrics metrics = null;
    private Font font = new Font("Monospaced", Font.BOLD, 24);
    private Object textMonitor = new Object();
    private boolean doUpdate = true;

    public Ticker(String text, Color textColor, Color backColor) {
        super();
        this.text = text;
        setTextColor(textColor);
        setBackground(backColor);
        metrics = getFontMetrics(font);
        width = metrics.stringWidth(text);
        height = metrics.getHeight();
        yPos = height;
        int localWidth = (int) (width * 0.4);

        width = width > defaultWidth ? localWidth : defaultWidth;
        height = height + 10 > defaultHeight ? height + 10 : defaultHeight;
        setSize(new Dimension(width, height));
        setInitXPos();
        xPos = getSize().width / 2;
        start();
    }
}
```
public void setInitXPos() {
    xPos = getSize().width / 2;
}

public void setText(String _text) {
    synchronized (textMonitor) {
        if (doUpdate) {
            setInitXPos();
            text = _text;
            width = metrics.stringWidth(text);
            setDoUpdate(false);
        }
    }
}

public String getText() {
    synchronized (textMonitor) {
        return text;
    }
}

public void setDoUpdate(boolean doUpdate) {
    this.doUpdate = doUpdate;
}

public void setTextColor(Color textColor) {
    setForeground(textColor);
}

public void setBackgroundColor(Color backColor) {
    setBackground(backColor);
}

public void start() {
    if (runner == null) {
        runner = new Thread(this);
        runner.start();
    }
}

public void stop() {
    runner = null;
}

public void computeCoordinates() {
    if (xPos < -width) {
        setDoUpdate(true);
        xPos = getSize().width;
    } else {
        xPos -= 1;
    }
}
public void run() {
    while (runner != null) {
        computeCoordinates();
        repaint();
        try {
            Thread.sleep(25);
        } catch (InterruptedException ie) {
            System.out.println("Error" + ie);
        }
    }
}

public void paint(Graphics g) {
    String t = "A fierce salty wind came from the sea, covering an air " +
    "Scottish heath with a stinging dampness. Forks of lightening exposed"
    "how barren this place was. There was no sign of life anywhere. Then a"
    "cackle could be heard. Three witches appeared from thin air," +
    "hovering, barely visible, in the fog. They landed together at a spot"
    "were a cauldron stood, awaiting their arrival. The steam from the" +
    "witches’ brew rose in the night air and mixed with the fog. The" +
    "witches stood chanting in turns around their cauldron end end end";
    super.paint(g);
    g.drawString(text, xPos, yPos);
}

Source Code: Src/14/Ticker.java

• The driver class:

    public class TickerDemo extends JFrame implements ActionListener, ItemListener
    {
        private static int soManyTickers = 0;
        private String IPAddr = "233.66.77.99";
        private Ticker ticker;
        private JPanel panel = new JPanel();
        private JLabel fore, back;
        private JButton start, stop;
        private String newsIPAddr[] = { "233.66.77.99", "233.66.77.101" };
        private final int newsConst = 0;
        private final int emergencyConst = 1;
private String actService = newsService[0];

private Color colors[] = {Color.orange, Color.pink,
    Color.cyan, Color.magenta,
    Color.yellow, Color.black,
    Color.white, Color.gray,
    Color.lightGray, Color.darkGray,
    Color.red, Color.green, Color.blue};

private String colorNames[] = {"Orange", "Pink",
    "Cyan", "Magenta",
    "Yellow", "Black",
    "White", "Gray",
    "Light Gray", "Dark Gray",
    "Red", "Green", "Blue"};

private Choice newsChoice, textChoice, backChoice;
private Color foreground, background;

public static void main(String args[]) {
    makeOne("233.66.77.99");
}

public TickerDemo() {
    super();
}

public TickerDemo(String title) {
    super(title);
}

public static void makeOne(String IPAddr) {
    TickerDemo aTD = new TickerDemo("Version 2.0");
    soManyTickers ++;
    aTD.IPAddr = IPAddr;
    aTD.init();
    aTD.pack();
    aTD.setResizable(false);
    aTD.show();
    aTD.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            soManyTickers --;
            if (soManyTickers == 0)
                System.exit(0);
        }
    });
}

public void init() {
    GridLayout aLayoutManager = new GridLayout(2, 1);
    foreground = Color.yellow;
    background = Color.black;
ticker = new Ticker("Waiting for a message ....", foreground, background);
start = new JButton("Start");
start.addActionListener(this);
stop = new JButton("Stop");
stop.addActionListener(this);
fore = new JLabel(" Text Color:");
back = new JLabel(" Background Color:");
newsChoice = new Choice();
for (int i = 0; i < newsService.length; i++)
    newsChoice.add(newsService[i]);
newsChoice.addItemListener(this);
if (IPAddr.equals(newsIPAddr[0]))
    newsChoice.select(0);
else
    newsChoice.select(1);

textChoice = new Choice();
backChoice = new Choice();
for (int i = 0; i < colorNames.length; i++)
    {
        textChoice.add(colorNames[i]);
        backChoice.add(colorNames[i]);
    }
textChoice.addItemListener(this);
backChoice.addItemListener(this);
textChoice.select(4);
backChoice.select(5);
panel.setLayout(new FlowLayout());
panel.add(start);
panel.add(stop);
panel.add(fore);
panel.add(textChoice);
panel.add(back);
panel.add(backChoice);
panel.add(newsChoice);
panel.setBackground(Color.lightGray);
getContentPane().setLayout(aLayoutManager);
getContentPane().add("North", ticker);
getContentPane().add("South", panel);

public void actionPerformed(ActionEvent ae)
{
    if (ae.getSource() == start)
        ticker.start();
    else if (ae.getSource() == stop)
        ticker.stop();
}

public void itemStateChanged(ItemEvent ie)
if (ie.getSource() == newsChoice) {
    if ((newsService[0].equals(newsService[newsChoice.getSelectedIndex()])) ||
        (newsService[1].equals(newsService[newsChoice.getSelectedIndex()])) ) {
        actService = newsService[newsChoice.getSelectedIndex()];
        ticker.stop();
        ticker.setDoUpDate(true);
        ticker.setText("Waiting for a message ....");
        ticker.start();
        oldService = newsChoice.getSelectedIndex();
    }
    if (newsService[2].equals(
        newsService[newsChoice.getSelectedIndex()])) {
        makeOne(newsIPAddr[0]);
        newsChoice.select(oldService);
    }
    if (newsService[3].equals(
        newsService[newsChoice.getSelectedIndex()])) {
        makeOne(newsIPAddr[1]);
        newsChoice.select(oldService);
    } else {
        ticker.setBackground(colors[backChoice.getSelectedIndex()]);
        ticker.setTextColor(colors[textChoice.getSelectedIndex()]);
        repaint();
    }
}

Source Code: Src/14/TickerDemo.java

19. Under Construction
20. Under Construction
21. Under Construction
22. Under Construction
23. Homework 1 - Intersession 2153

Posted: December/29/2015
Due: Jan/6/2016 8:00am

Use glados.cs.rit.edu, if kansas.cs.rit.edu does not work.
23.1. Homework 1 - Intersession 2153.1 (10 Points)

Use a recursive algorithm to determine the gcd of 10 numbers. You can store the numbers in an array.

Example:

```
% java GCD 2 4 6 8 10 12 14 16 18 20
2
```

Submit your solution via:

```
% ssh kansas.cs.rit.edu
# password
# go to the directory where your solution is ...
try hpb-grd lab15-1 "all files you need to compile"
```

23.2. Homework 2 - Intersession 2153.1 (10 Points)

Find all prime numbers between 1 and 2 * Integer.MAX_VALUE. For this hw you should assume Integer.MAX_VALUE = 100000. You have to use the idea of a BitSet implementation, but you can not use any class outside the basic types and array.

Example:

```
% java Prime
1 2 3 5 ...
```

Submit your solution via:

```
% ssh kansas.cs.rit.edu
# password
# go to the directory where your solution is ...
try hpb-grd lab15-2 "all files you need to compile"
```

23.3. Homework 3 - Intersession 2153.1 (10 Points)

Find out all possible combination of the numbers between 1 and 10 with a recursive and an iterative algorithm.

Example:

```
% java Prime
1 2 3 5 ...
```

Submit your solution via:

```
% ssh kansas.cs.rit.edu
# password
# go to the directory where your solution is ...
try hpb-grd lab15-3 "all files you need to compile"
```

24. Homework 2 - Intersession 2153

Posted: Jan/7/2015
Due: Jan/11/2016 8:00am

Use glados.cs.rit.edu, if kansas.cs.rit.edu does not work.
24.1. Homework 1 - Intersession 2153.1 (10 Points)

Explain the behavior of the following code and answer the question asked in the code.

```java
public class S1 {
    public static int staticV = 0;
    public int instanceV = 1;

    public static void printsStatic() {
        System.out.println("S1: staticV: " + staticV);
    }

    public void printInstance() {
        System.out.println("S1: instanceV: " + instanceV);
    }
}
```

```java
public class S2 extends S1 {
    public static int staticV = 2;
    public int instanceV = 5;

    public static void printsStatic() {
        System.out.println("S2: staticV: " + staticV);
    }

    public void printInstance() {
        System.out.println("S2: instanceV: " + instanceV);
    }
}
```

```java
public static void main(String args[]) {
    S1 aS1 = new S1();
    aS1.printInstance();
    aS1.printsStatic();
    S2 aS2 = new S2();
    aS2.printInstance();
    aS2.printsStatic();
    aS1 = (S1)aS2;
    aS1.printInstance();
    aS1.printsStatic();
}
```

Submit your solution via:
24.2. Homework 2 - Intersession 2153.1 (10 Points)
Create a problem statement which can best be solved with abstract classes. Implement a solution.
Submit your solution via:

% ssh kansas.cs.rit.edu
# password
# go to the directory where your solution is ...
try hpb-grd lab2-1 "all files you need to compile"

24.3. Homework 3 - Intersession 2153.1 (10 Points)
Create a problem statement which can best be solved with abstract classes. Implement a solution.

% ssh kansas.cs.rit.edu
# password
# go to the directory where your solution is ...
try hpb-grd lab2-2 "all files you need to compile"

25. Homework 3 - Intersession 2153

Posted: Jan/8/2015
Due: Jan/13/2016 8:00am

Use glados.cs.rit.edu, if kansas.cs.rit.edu does not work.

25.1. Homework 1 - Intersession 2153.1 (10 Points)
Implement (a) class(es) which do(es) the following:
• Writing: Write the data into a compressed stream (x) and sign the file
• Reading: Read the file and ensure the file is correct. The API is up to you.
Implement also a test environment.
Submit your solution via:

% ssh kansas.cs.rit.edu
# password
# go to the directory where your solution is ...
try hpb-grd lab2-1 "all files you need to compile"

25.2. Homework 2 - Intersession 2153.1 (10 Points)
Write a class which serializes and reads 2Dimensional matrices. The matrix will only be written or read, if the matrix is correct.
Implement also a test environment.
Submit your solution via:
% ssh kansas.cs.rit.edu
# password
# go to the directory where your solution is ...
try hpb-grd lab2-2 "all files you need to compile"

25.3. Homework 3 - Intersession 2153.1 (10 Points)
Write a program which finds the prime number representation using a parallel algorithm.

% ssh kansas.cs.rit.edu
# password
# go to the directory where your solution is ...
try hpb-grd lab2-3 "all files you need to compile"

26. Homework 4 - Intersession 2153

Posted: Jan/13/2015
Due: Jan/18/2016 8:00am

Use glados.cs.rit.edu, if kansas.cs.rit.edu does not work.

26.1. Homework 1 - Intersession 2153.1 (10 Points)
Implement a program which creates 3 threads. Each thread can print its id. Arrange it in such a way that the output is: 1 2 3 1 2 3 1 2 3 ...

26.2. Homework 2 - Intersession 2153.1 (10 Points)
Write a program which simulates musical chairs. Each child is represented by one thread.

27. Homework 5 - Intersession 2153

Posted: Jan/18/2015
Due: Jan/20/2016 8:00am

Use glados.cs.rit.edu, if kansas.cs.rit.edu does not work.

27.1. Homework 1 - Intersession 2153.1 (10 Points)
Implement a program which always ends up in a deadlock. You can only use wait, notify and synchronize.

27.2. Homework 2 - Intersession 2153.1 (10 Points)
Write a program which has n producers and k consumers. Producer k can only produce k items at a time, and consumer k can only consume k items at a time. The storage area is 2 * n.

28. Homework 6 - Intersession 2153

Posted: Jan/18/2015
Due: Never

28.1. Homework 1 - Intersession 2153.1 (10 Points)
Implement a client/server environment with the following functionality:
• The client sends two numbers \((a, b)\) to the server
• The server reads the number, and returns \(a+b\).

You have implemented the solutions for 3 different communication mechanisms:
• TCP/IP
• UDP
• RMI

29. Examination I—Jan/13/2016

Name: 

Grader: 

Time: 60 Minutes
Total Points: 56

I will grade your answer as wrong, if I cannot read what you wrote.
Question 1 (8)

When would you use an Interface and when an Abstract Class?
Question 2 (8)

What are all the possible outputs?

```java
public class T_1 extends Thread {
    private int info;
    static int x = 0;

    public T_1 (int info) {
        this.info = info;
    }

    public void run () {
        if ( info == 1 )
            x = 3;
        else if ( info == 0 )
            x = 2;
        else
            x = 1;
    }

    public static void main (String args []) {
        T_1 aT1 = new T_1(1);
        T_1 aT2 = new T_1(2);
        aT1.start();
        aT2.start();
        System.out.println(x);
    }
}
```

Source Code: Src/81/T_1.java

Answer:

Explain your answer for each output:
Question 3 (8)
Will the below code compile or not? Explain each marked line.

```java
1 import java.util.*;
2
3 public class M<E, V> { // 1
4     List<E> data = new ArrayList<E>(); // 2
5     ArrayList<E> data2 = new ArrayList<E>(); // 3
6     Vector<V> volume = new Vector<V>(); // 4
7
8     public V getV() { // 5
9         return volume.elementAt(0); // 6
10     }
11 }
12
13 class MyVector<E> { // 7
14     E[] data = null;
15     public MyVector( int size ) { // 8
16         data = new E[size];
17     }
18 }
19
Source Code: Src/81/M.java
```

1.

2.

3.

4.

5.

6.
7.

8.

9.
Question 4 (8)
Add code to execute `sample()` and `sample2()`.

```java
public class OuterClass {
    public void sample() {
        System.out.println("In outer class");
    }

    public static class InnerClass {
        public void sample2() {
            System.out.println("In inner class");
        }
    }

    public static void main(String args[]) {
        OuterClass obj = new OuterClass();
    }
}
```

Source Code: Src/81/OuterClass.java
Question 5 (8)

Is

-->

<--

-->

<--

The only possible output?

```java
public class T_2 extends Thread {
    private String info;

    public T_2(String info) {
        this.info = info;
    }

    private void inProtected() {
        synchronized (info) {
            System.err.println("-->");
            try {
                sleep(1000);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        System.err.println("<-- " + info);
    }

    public void run() {
        inProtected();
    }

    public static void main(String args[]) {
        new T_2("hello").start();
        new T_2("hello").start();
    }

    Source Code: Src/81/T_2.java
}
```

Explain your answer.
Question 6 (8)
Is ’0 1 0 1 0 1...” the only possible output?

```java
public class T extends Thread {
    private String info;
    static Object o = new Object();
    public T (String info) {
        this.info = info;
    }
    public void run () {
        synchronized ( o ) {
            while ( true ) {
                System.out.println(info);
                try {
                    o.notify();
                    sleep(100);
                    o.wait();
                } catch ( Exception e ) { }
            }
        }
    }
    public static void main (String args []) {
        new T("0").start();
        ( new T("1") ).start();
    }
}
Source Code: Src/81/T.java
```

Explain your answer.